Networking protocols and administration

ITV8030

lecture 4: tcp control, icmp, routing, ...

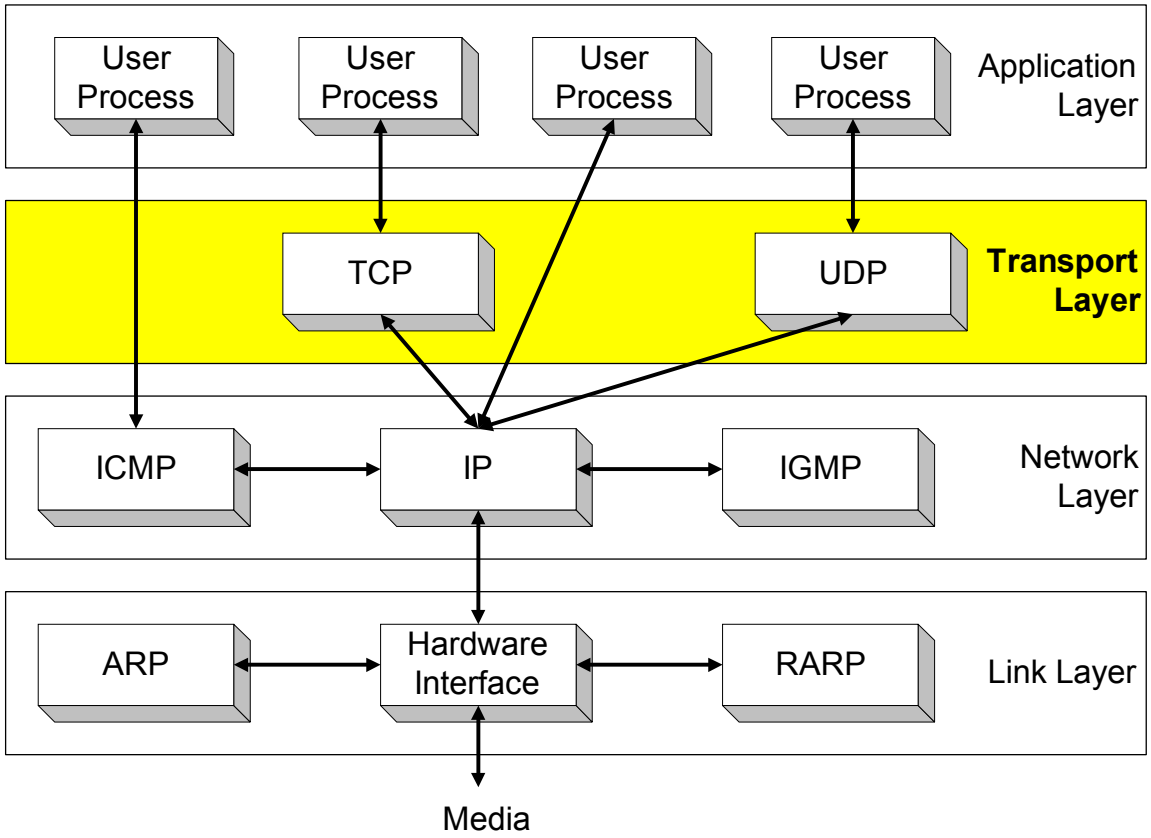
lecture plan

- Refresher: TCP protocol main parts
- TCP control:
  - Flow, error, congestion
- Back to IP level and surroundings:
  - ICMP
  - Routers and routing principles

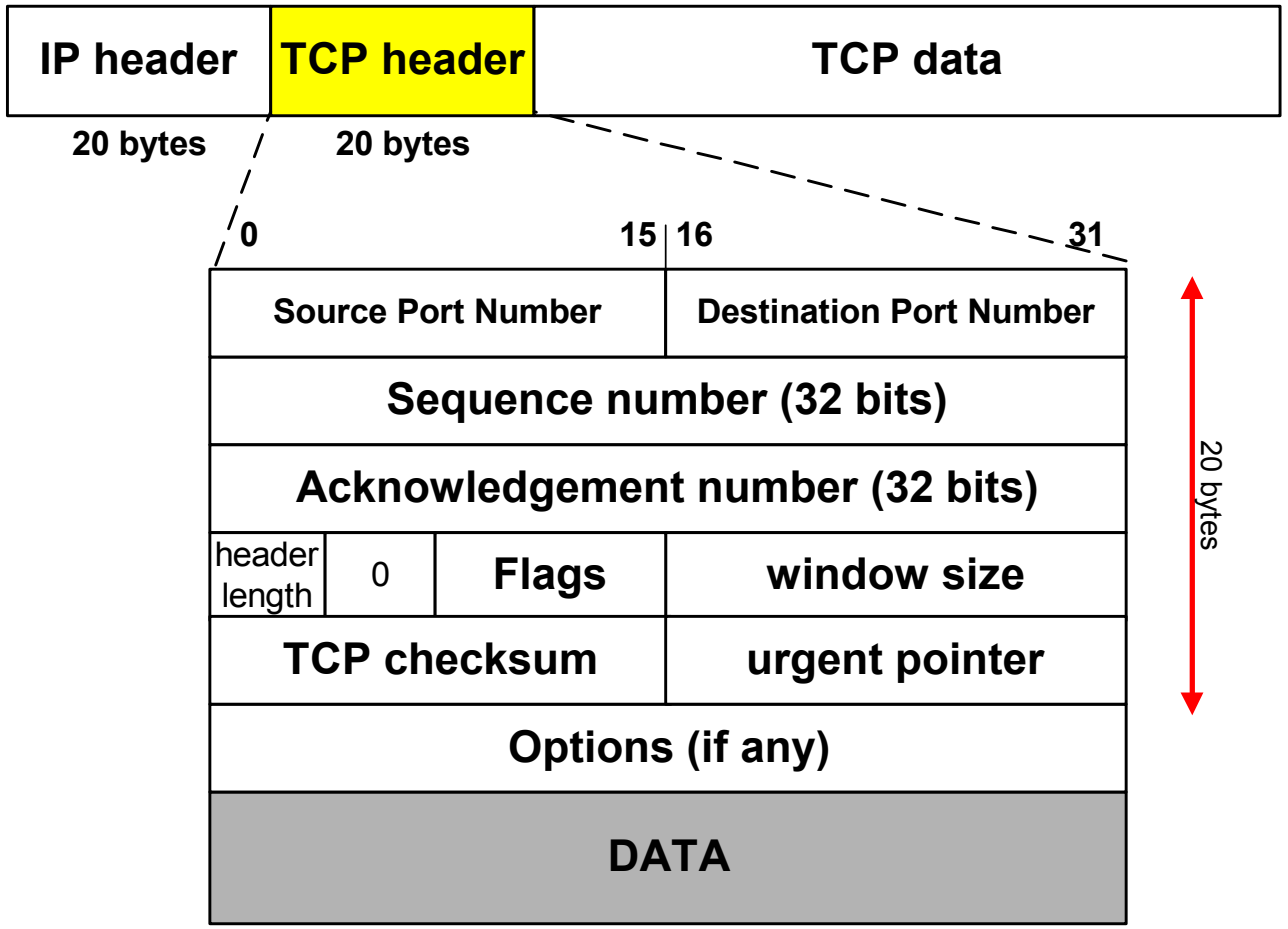
- Using mostly slides from Univ of Virginia / Univ of Toronto

Orientation

- We move one layer up and look at the transport layer.



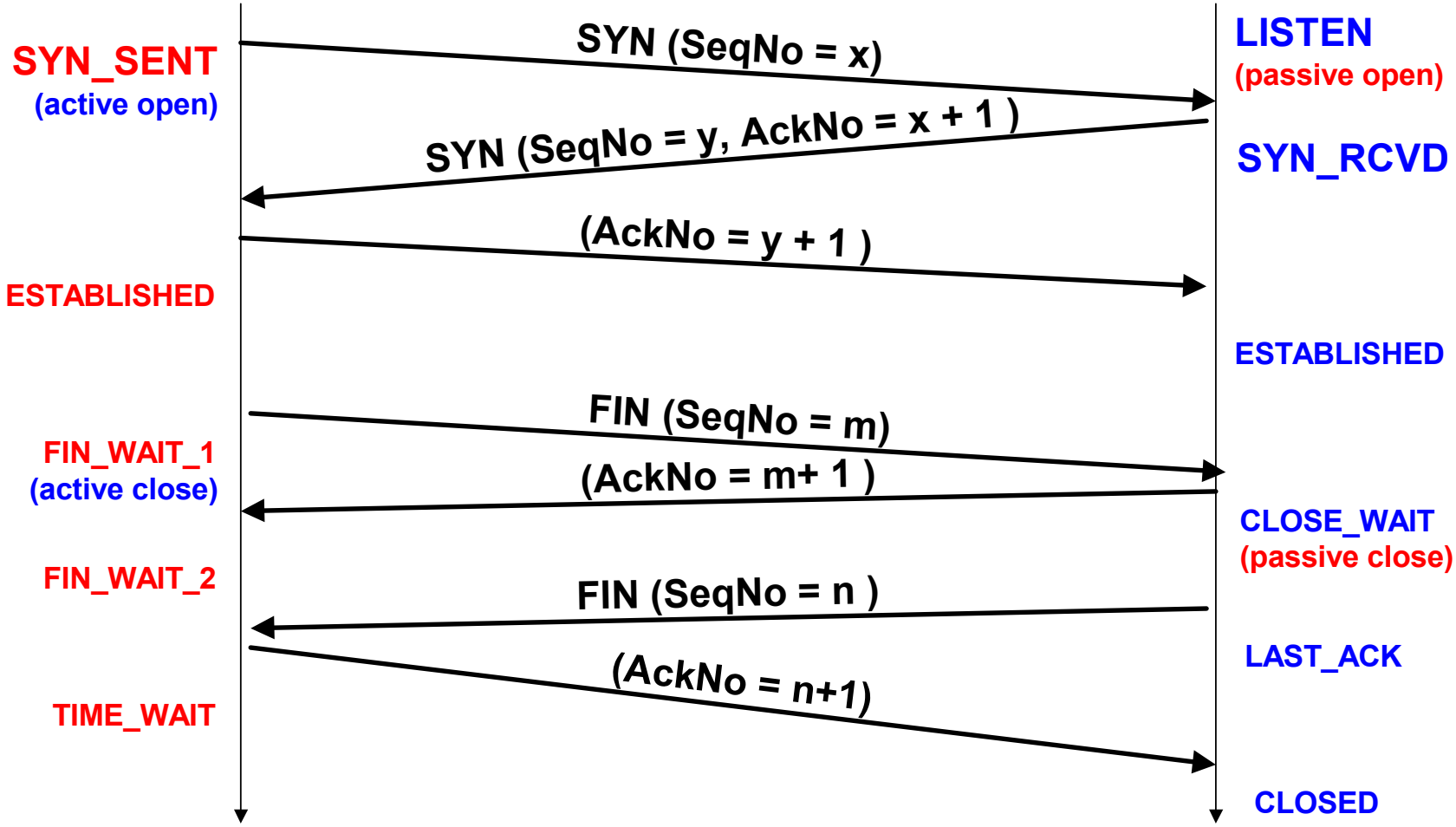
TCP Format


IP header
TCP header
TCP data
20 bytes
20 bytes
0
15
16
31
Source Port Number
Destination Port Number
Sequence number (32 bits)
Acknowledgement number (32 bits)
header length
0
Flags
window size
TCP checksum
urgent pointer
Options (if any)
DATA
20 bytes

How to understand TCP

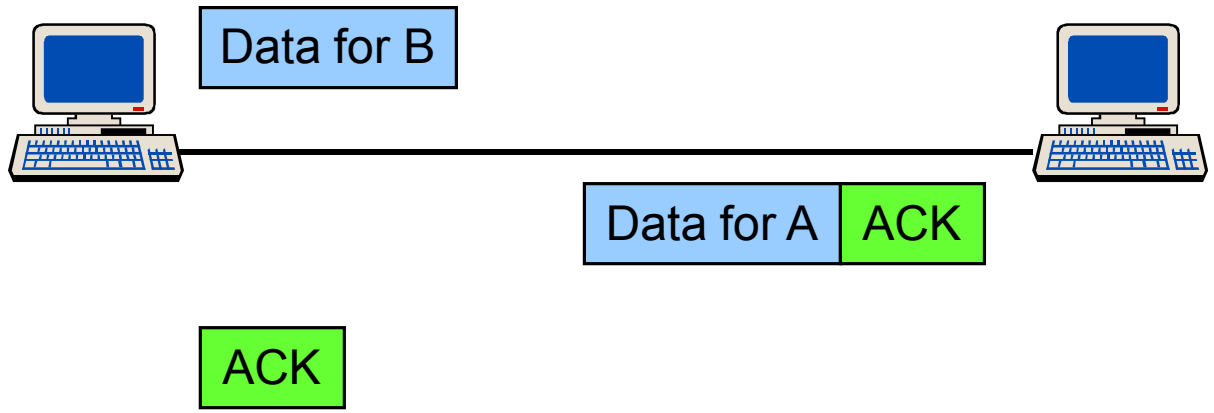
- Think like this:
  - **Core TCP**: protocol parts that must be implemented
  - **Optional** TCP protocol parts:
    - some implemented in sender/receiver, some not
    - sender and receiver advertise available options
    - important ones normally implemented
  - **Timing, packet size, etc options**:
    - main goal: better bandwidth usage, speedier operation
    - sender may determine at will
    - sender algorithms may vary

TCP States in “Normal” Connection Lifetime



Acknowledgements in TCP

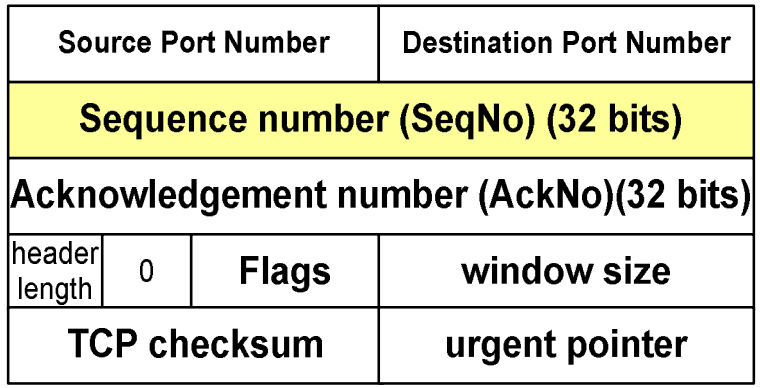
- TCP receivers use acknowledgments (ACKs) to confirm the receipt of data to the sender
- Acknowledgment can be added ("piggybacked") to a data segment that carries data in the opposite direction
- ACK information is included in the the TCP header
- Acknowledgements are used for flow control, error control, and congestion control



Sequence Numbers and Acknowledgments in TCP

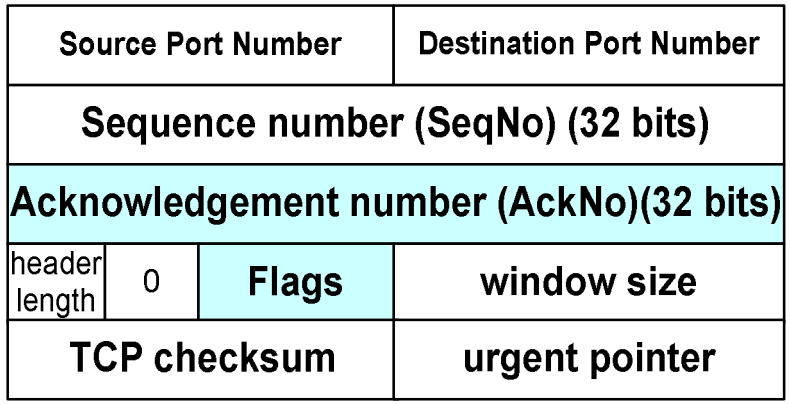
- TCP uses sequence numbers to keep track of transmitted and acknowledged data
- Each transmitted byte of payload data is associated with a sequence number
- **Sequence numbers count bytes and not segments**
- Sequence number of first byte in payload is written in *SeqNo* field
- Sequence numbers wrap when they reach 2^{32}-1

- The sequence number of the first sequence number (Initial sequence number) is negotiated during connection setup



Sequence Numbers and Acknowledgments in TCP

- An acknowledgment is a confirmation of delivery of data
- When a TCP receiver wants to acknowledge data, it
  - writes a sequence number in the AckNo field, and
  - sets the ACK flag



IMPORTANT: An acknowledgment confirms receipt for all unacknowledged data that has a smaller sequence number than given in the AckNo field

Example: AckNo=5 confirms delivery for 1,2,3,4 (but not 5).

Cumulative Acknowledgements

- With cumulative ACKs, the receiver can only acknowledge a segment if all previous segments have been received

- With cumulative ACKs, receiver cannot selectively acknowledge blocks of segments:
  e.g., ACK for S_0-S_3 and S_5-S_7 (but not for S_4)

- Note: The use of cumulative ACKs imposes constraints on the retransmission schemes:
  - In case of an error, the sender may need to retransmit all data that has not been acknowledged

Rules for sending Acknowledgments

- TCP has rules that influence the transmission of acknowledgments

- Rule 1: Delayed Acknowledgments
  - *Goal:* Avoid sending ACK segments that do not carry data
  - *Implementation:* Delay the transmission of (some) ACKs

- Rule 2: Nagle's rule
  - *Goal:* Reduce transmission of small segments
    Implementation: A sender cannot send multiple segments with a 1-byte payload (i.e., it must wait for an ACK)

TCP Connection Termination

- Each end of the data flow must be shut down independently **("half-close")**
- If one end is done it sends a FIN segment. This means that no more data will be sent

- Four steps involved:

  (1) X sends a FIN to Y **(active close)**

  (2) Y ACKs the FIN,

  (at this time: Y can still send data to X)

  (3) and Y sends a FIN to X **(passive close)**

  (4) X ACKs the FIN.

TCP Flow Control

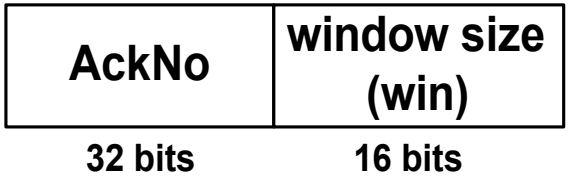
TCP Flow Control

- TCP uses a version of the **sliding window flow control,** where
  - Sending acknowledgements is separated from setting the window size at sender
  - Acknowledgements do not automatically increase the window size

- During connection establishment, both ends of a TCP connection set the initial size of the sliding window

Window Management in TCP

- The receiver is returning two parameters to the sender



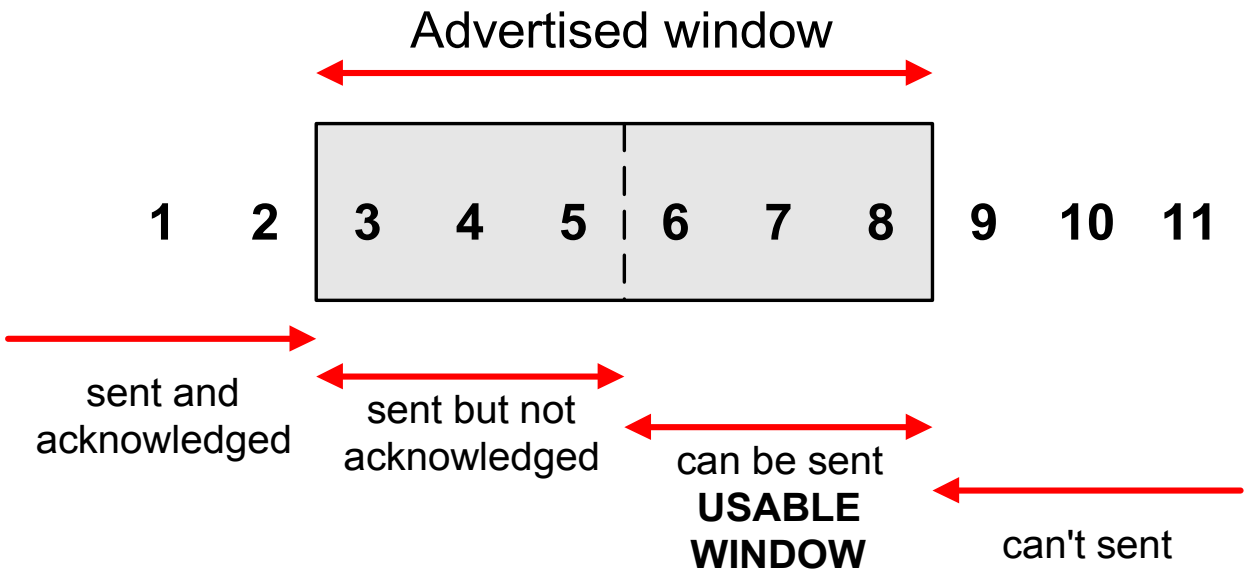
- The interpretation is:
  - **I am ready to receive new data with**

    SeqNo= AckNo, AckNo+1, …., AckNo+Win-1

- Receiver can acknowledge data without opening the window
- Receiver can change the window size without acknowledging data

Sliding Window Flow Control

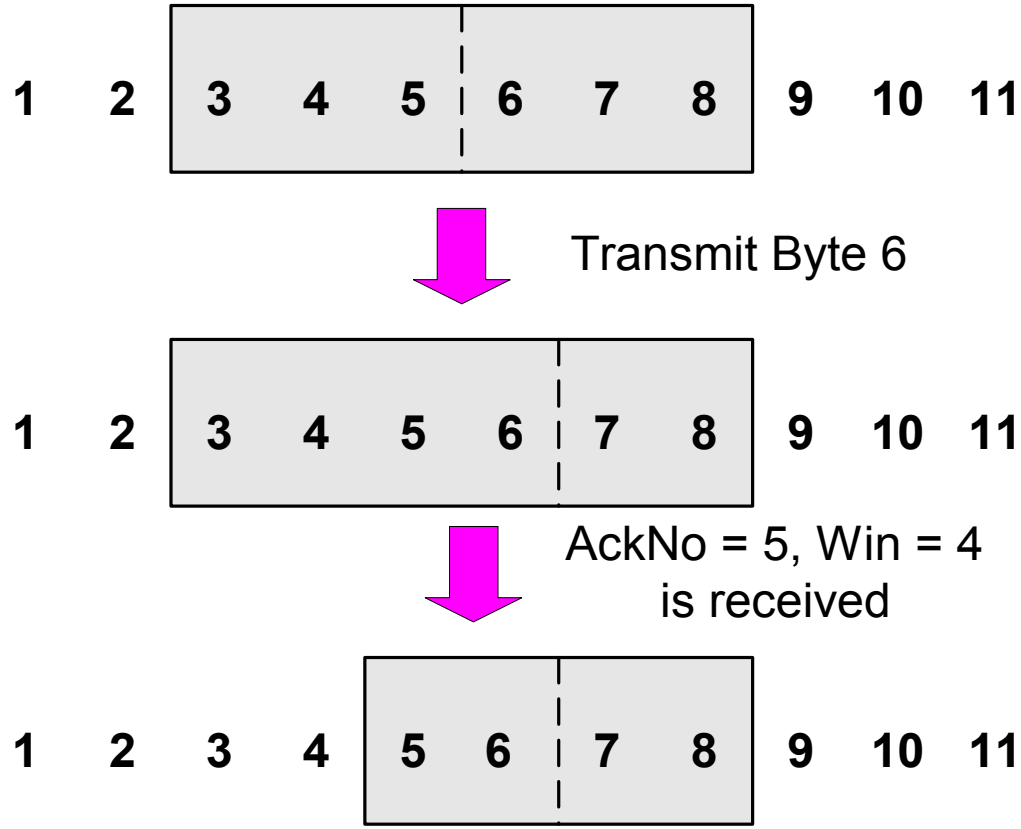
- Sliding Window Protocol is performed at the byte level:



- Here: Sender can transmit sequence numbers 6,7,8.

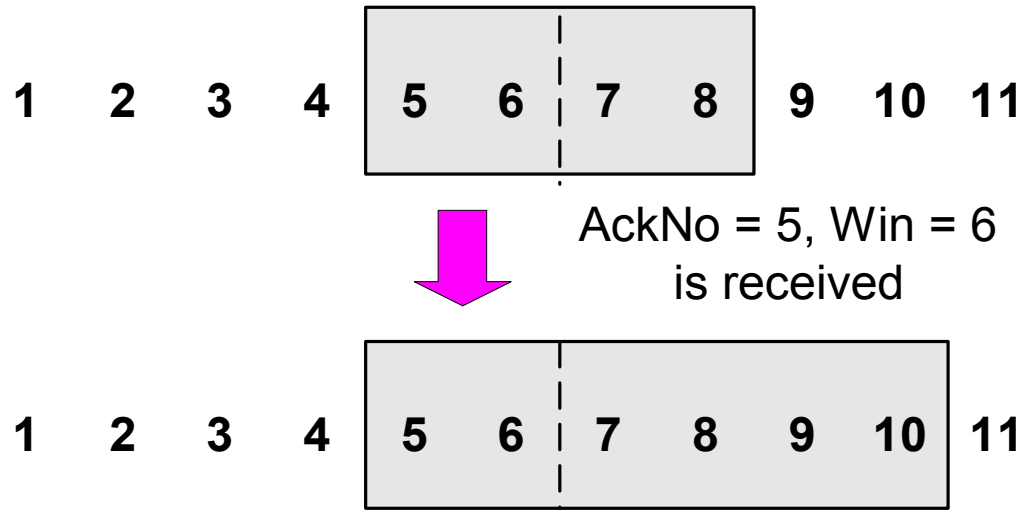
Sliding Window: "Window Closes"

• Transmission of a single byte (with SeqNo = 6) and acknowledgement is received



Sliding Window: "Window Opens"

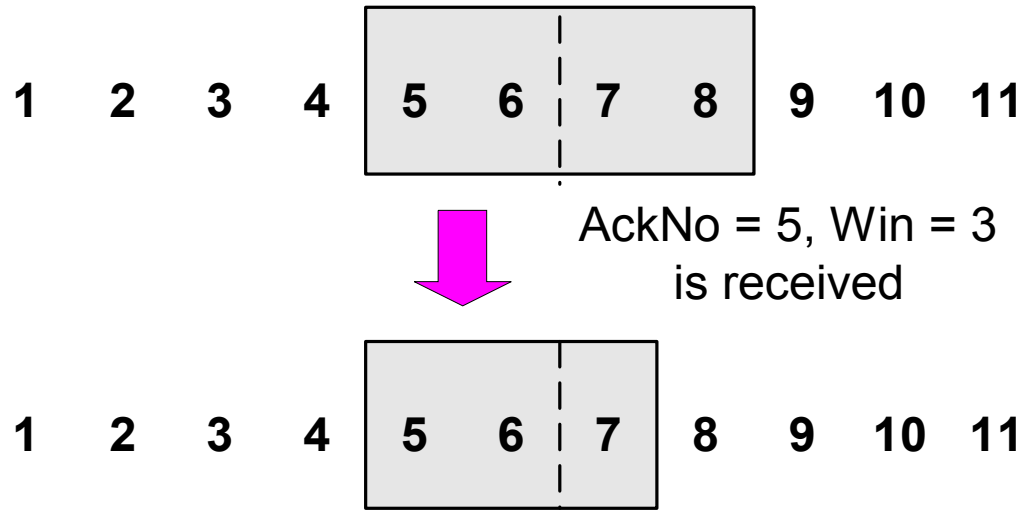
• Acknowledgement is received that enlarges the window to the right (AckNo



• A receiver opens a window when TCP buffer empties (meaning that data is delivered

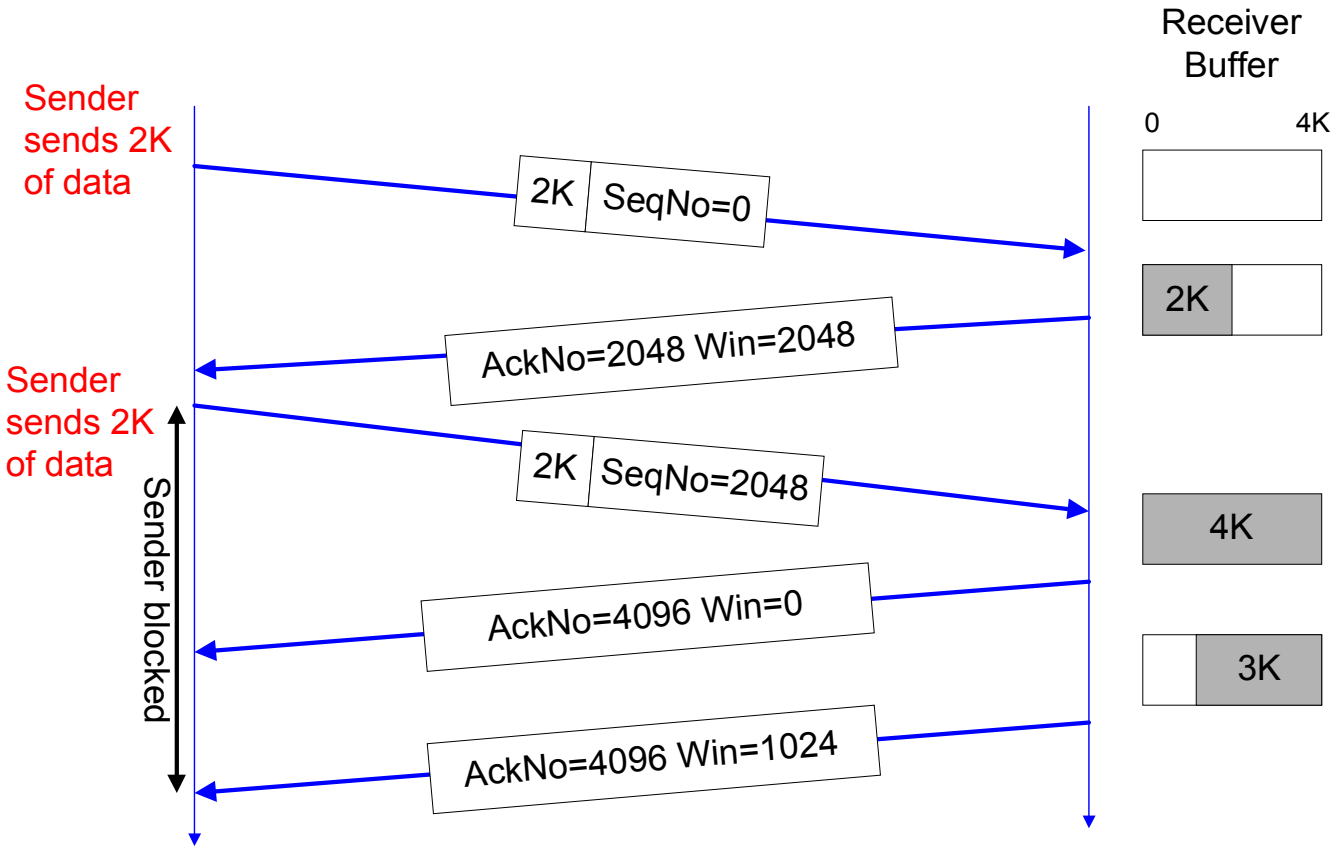
Sliding Window: "Window Shrinks"

• Acknowledgement is received that reduces the window from the right (AckNo



• Shrinking a window should not be used

Sliding Window: Example



TCP Error Control

Error Control in TCP

- TCP maintains a **Retransmission Timer** for each connection:
  - The timer is started during a transmission. A timeout causes a retransmission

- **TCP couples error control and congestion control** (i.e., it assumes that errors are caused by congestion)
  - Retransmission mechanism is part of congestion control algorithm

- **Here:** How to set the timeout value of the retransmission timer?

TCP Retransmission Timer

- **Retransmission Timer:**
  - The setting of the retransmission timer is crucial for efficiency
  - Timeout value too small → results in unnecessary retransmissions
  - Timeout value too large → long waiting time before a retransmission can be issued
  - A problem is that the delays in the network are not fixed
  - Therefore, the retransmission timers must be adaptive

Round-Trip Time Measurements

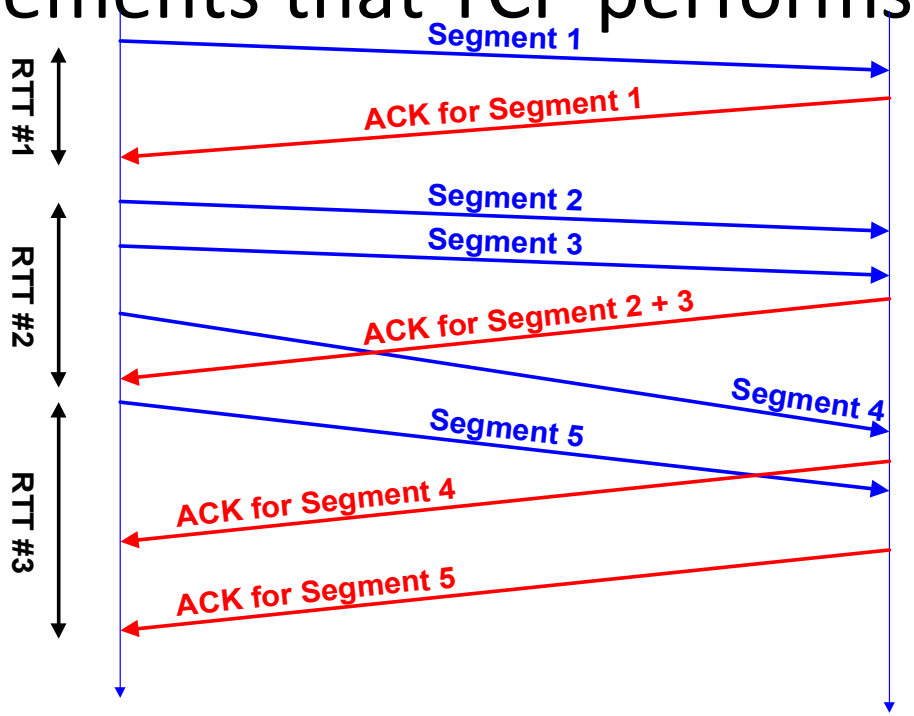
- The retransmission mechanism of TCP is adaptive
- The retransmission timers are set based on round-trip time (RTT) measurements that TCP performs

The RTT is based on time difference between segment transmission and ACK

But:

TCP does not ACK each segment

Each connection has only one timer



Round-Trip Time Measurements

- Retransmission timer is set to a **Retransmission Timeout (RTO) value**.
- RTO is calculated based on the *RTT* measurements.
- The RTT measurements are smoothed by the following estimators *srtt* and *rttvar*:

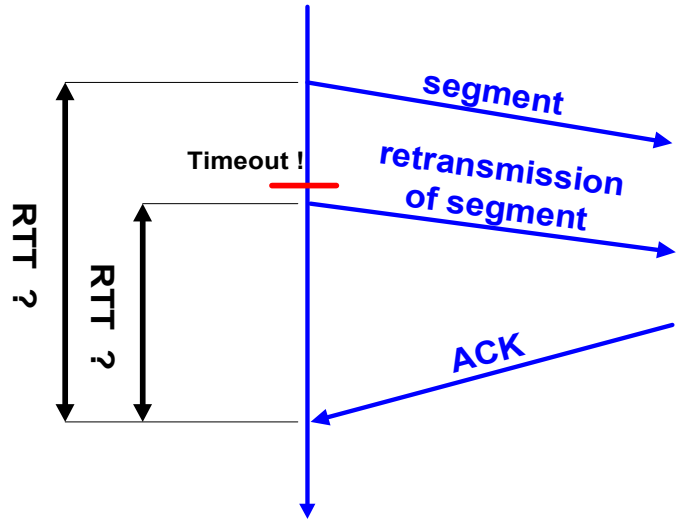
$$srtt_{n+1} = \alpha \ RTT \ + (1- \alpha) \ srtt_n$$

$$rttvar_{n+1} = \beta (\ | \ RTT - srtt_{n+1} \ | \ ) + (1- \beta) \ rttvar_n$$

$$RTO_{n+1} = srtt_{n+1} + 4 \ rttvar_{n+1}$$

Karn's Algorithm

- If an ACK for a retransmitted segment is received, the sender cannot tell if the ACK belongs to the original or the retransmission.

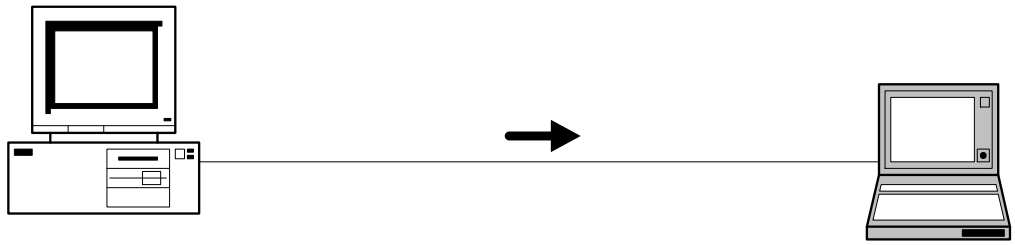


Karn's Algorithm:

Don't update srtt on any segments that have been retransmitted.
Each time when TCP retransmits, it sets:
$RTO_{n+1} = \max (2\ RTO_n, 64)$ (exponential backoff)

Measuring TCP Retransmission Timers

ftp session
from ellington
to satchmo



ellington.cs.virginia.edu *satchmo.cs.virginia.edu*

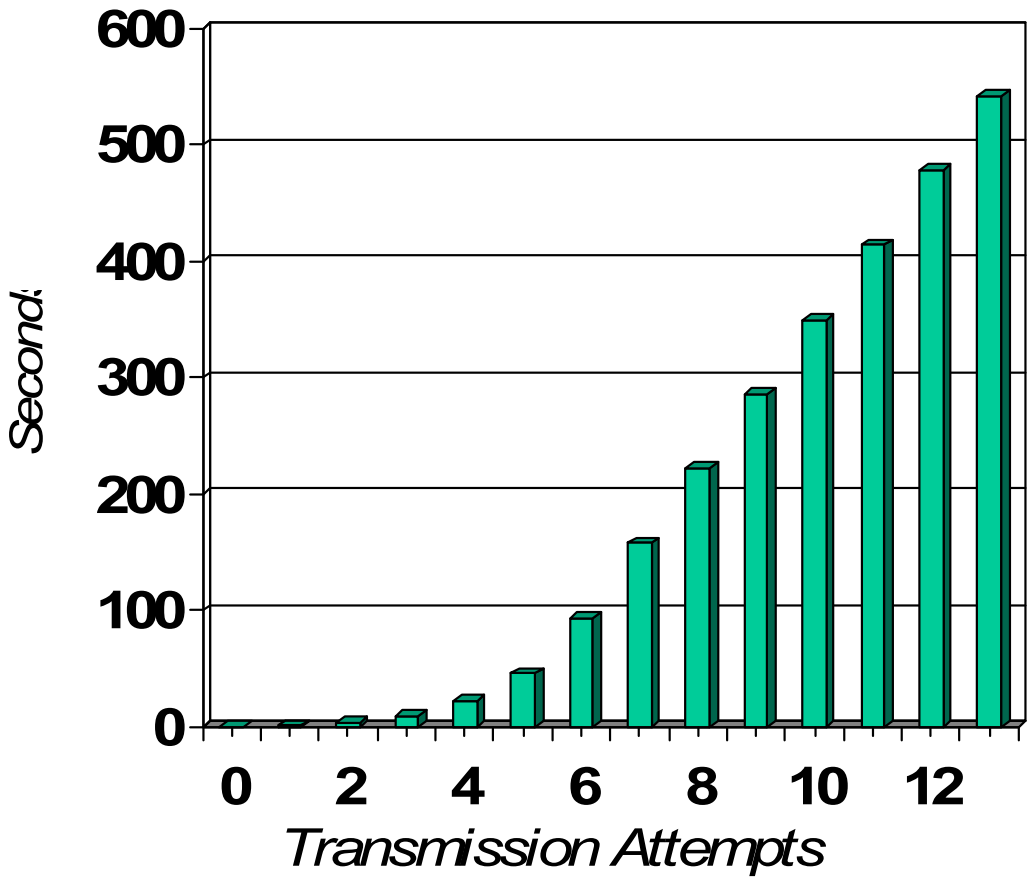
•Transfer file from ellington to satchmo

• Unplug Ethernet cable in the middle of file transfer

Exponential Backoff

- Scenario: File transfer between two machines. Disconnect cable.
- The interval between retransmission attempts in seconds is:

  1.03, 3, 6, 12, 24, 48, 64, 64, 64, 64, 64, 64, 64.
- Time between retrans-missions is doubled each time (Exponential Backoff Algorithm)
- Timer is not increased beyond 64 seconds
- TCP gives up after 13th attempt and 9 minutes.



TCP Congestion Control

TCP Congestion Control

- TCP has a mechanism for congestion control. The mechanism is implemented at the sender
- The window size at the sender is set as follows:

Send Window = MIN (flow control window, congestion window)

where

- flow control window is advertised by the receiver
- congestion window is adjusted based on feedback from the network

TCP Congestion Control

- TCP congestion control is governed by two parameters:
  - **Congestion Window** (**cwnd**)
  - **Slow-start threshhold Value** (**ssthresh)**

    Initial value is 2^{16}-1
- Congestion control works in <u>two modes</u>:
  - **slow start** (cwnd < ssthresh)
  - **congestion avoidance** (cwnd ≥ ssthresh

Slow Start

- Initial value: Set **cwnd = 1**
  - Note: Unit is a segment size. TCP actually is based on bytes and increments by 1 MSS (maximum segment size)

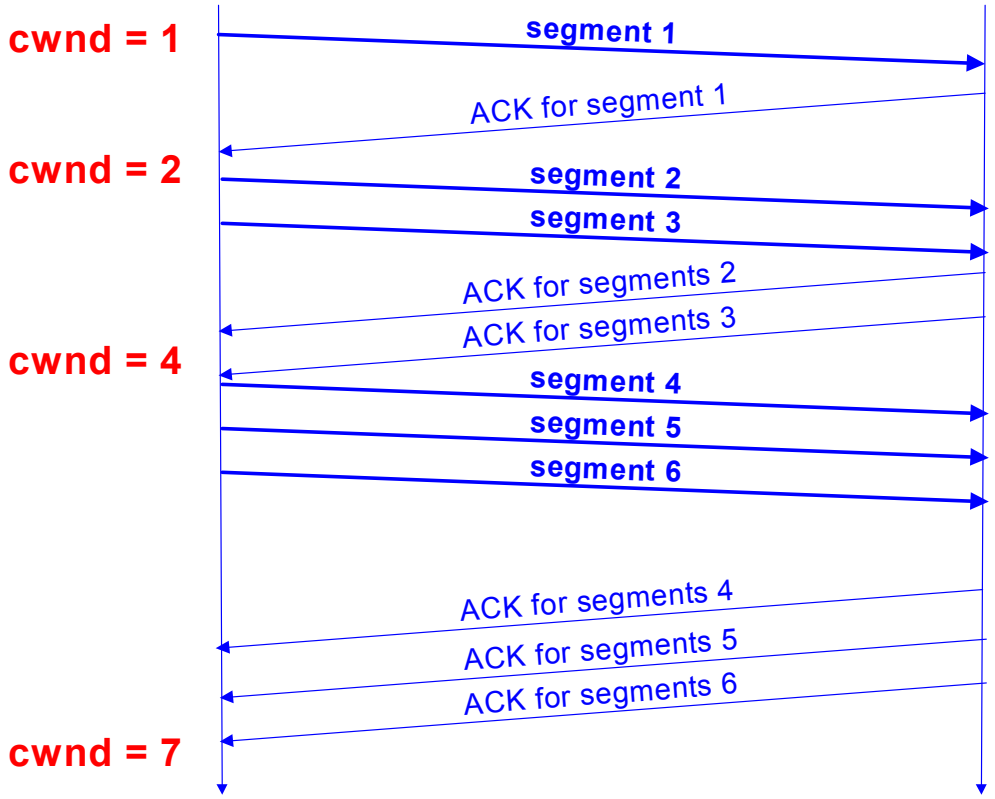
- The receiver sends an acknowledgement (ACK) for each Segment
  - Note: Generally, a TCP receiver sends an ACK for every other segment.
- Each time an ACK is received by the sender, the congestion window is increased by 1 segment:

  cwnd = cwnd + 1

  - If an ACK acknowledges two segments, cwnd is still increased by only 1 segment.
  - Even if ACK acknowledges a segment that is smaller than MSS bytes long, cwnd is increased by 1.
- Does Slow Start increment slowly? Not really.
  In fact, the increase of cwnd is exponential

Slow Start Example

- The congestion window size grows very rapidly
  - For every ACK, we increase cwnd by 1 irrespective of the number of segments ACK'ed
- TCP slows down the increase of *cwnd* when ***cwnd > ssthresh***



Congestion Avoidance

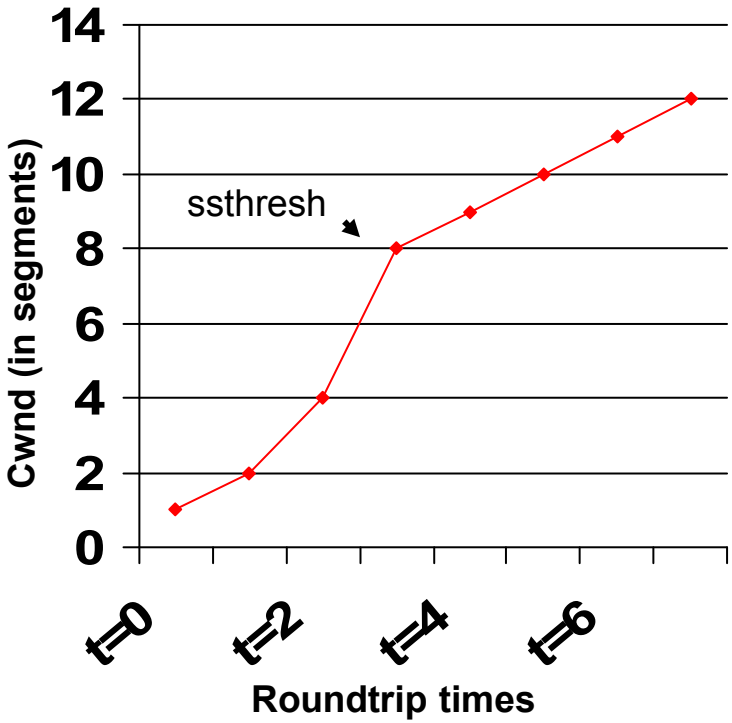
- Congestion avoidance phase is started if cwnd has reached the slow-start threshold value

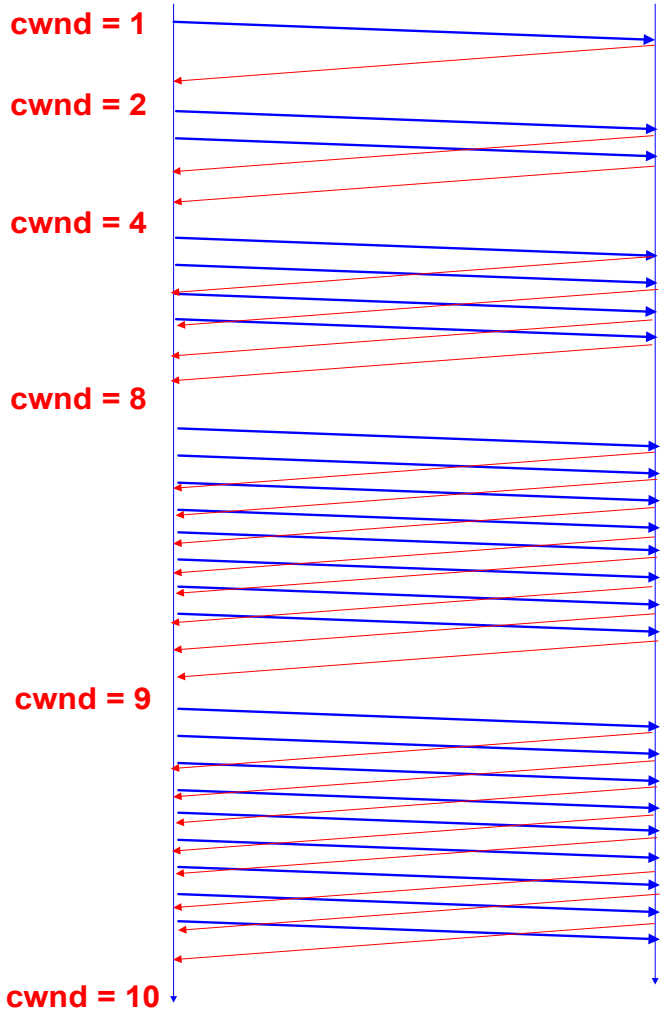
- If cwnd ≥ ssthresh then each time an ACK is received, increment cwnd as follows:
  - cwnd = cwnd + 1/ cwnd

- So *cwnd* is increased by one only if all *cwnd* segments have been acknowledged.

Example of Slow Start/Congestion Avoidance

Assume that *ssthresh = 8*





Responses to Congestion

- So, TCP assumes there is congestion if it detects a packet loss
- A TCP sender can detect lost packets via:
    - Timeout of a retransmission timer
    - Receipt of a duplicate ACK
- TCP interprets a Timeout as a binary congestion signal. When a timeout occurs, the sender performs:
  - cwnd is reset to one:

    cwnd = 1
  - ssthresh is set to half the current size of the congestion window:

    ssthressh = cwnd / 2
  - and slow-start is entered

Summary of TCP congestion control

Initially:

```
cwnd = 1;
ssthresh =
  advertised window size;
```

New Ack received:

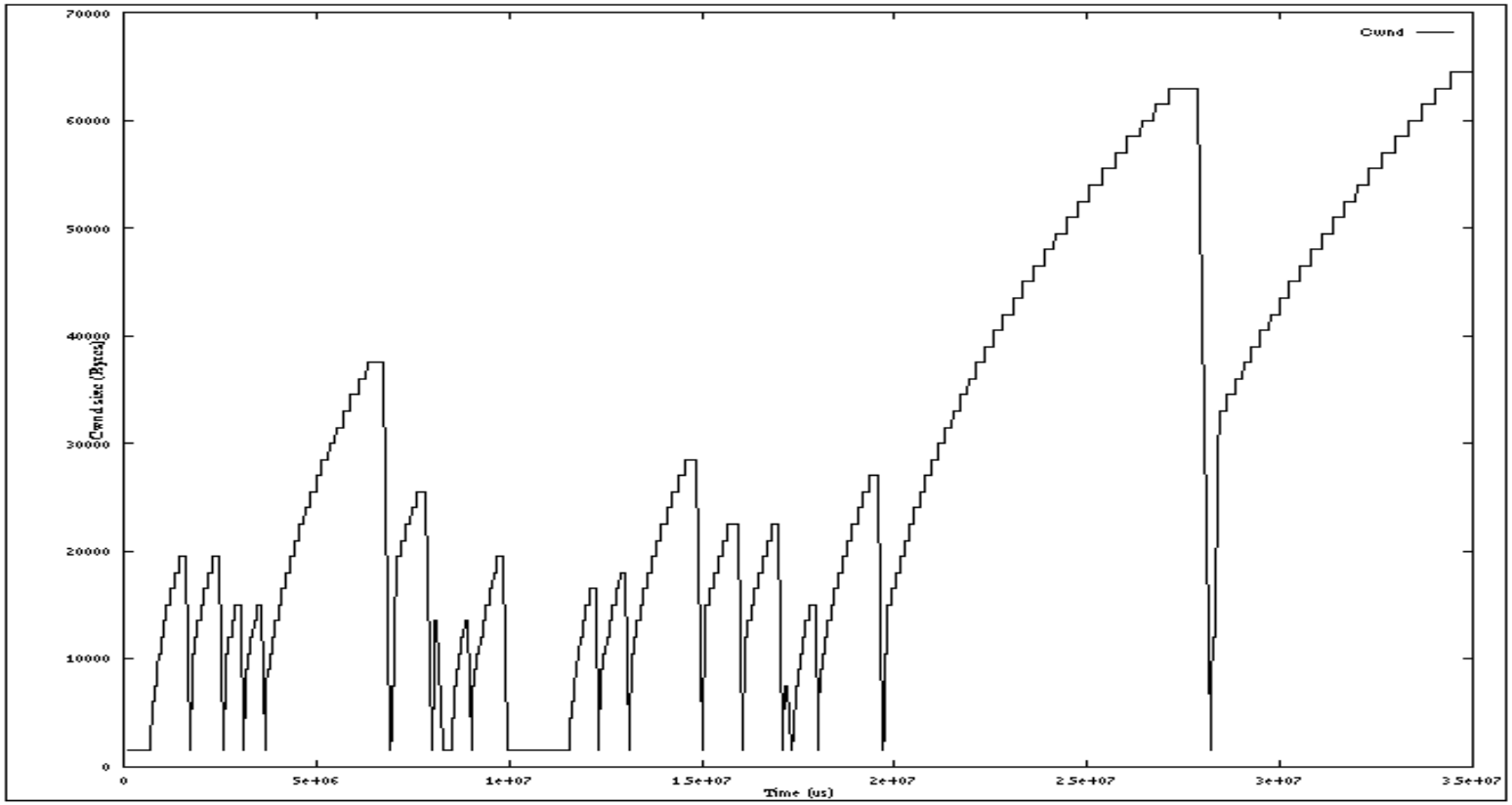
```
if (cwnd < ssthresh)
    /* Slow Start*/
    cwnd = cwnd + 1;
else
    /* Congestion Avoidance */
    cwnd = cwnd + 1/cwnd;
```

Timeout:

```
/* Multiplicative decrease */
ssthresh = cwnd/2;
cwnd = 1;
```

Slow Start / Congestion Avoidance

- A typical plot of cwnd for a TCP connection (MSS = 1500 bytes) with TCP Tahoe:

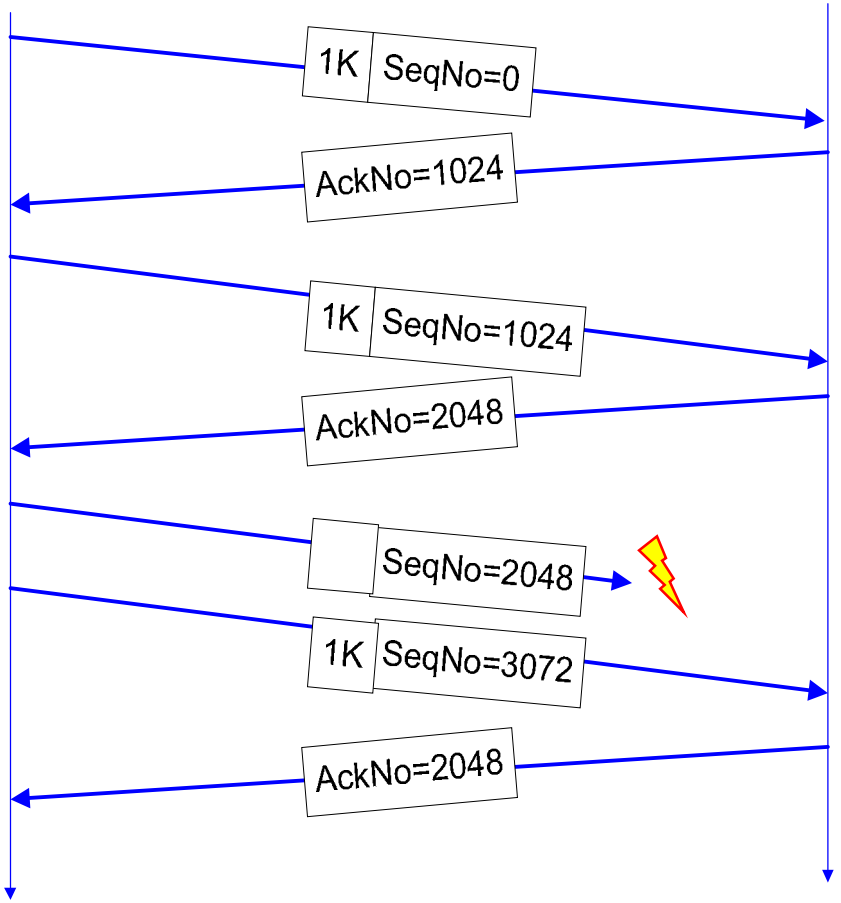


Flavors of TCP Congestion Control

- **TCP Tahoe** (1988, FreeBSD 4.3 Tahoe)
  - Slow Start
  - Congestion Avoidance
  - Fast Retransmit
- **TCP Reno** (1990, FreeBSD 4.3 Reno)
  - Fast Recovery
- **New Reno** (1996)
- **SACK** (1996)
- **RED** (Floyd and Jacobson 1993)

Acknowledgments in TCP

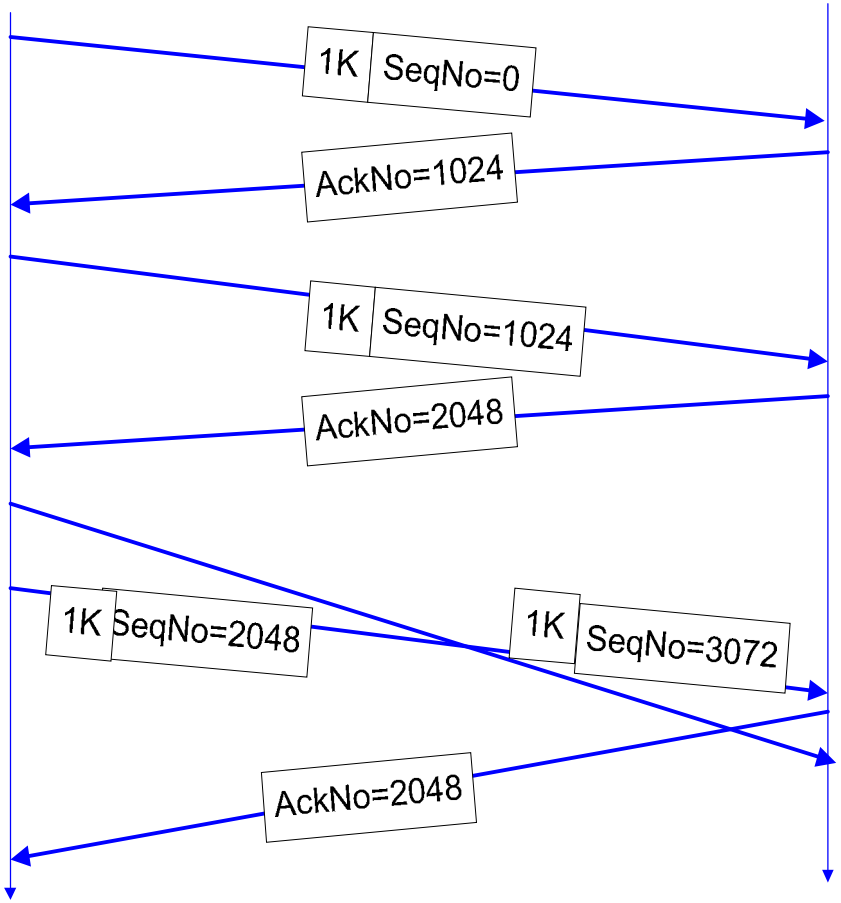
- Receiver sends ACK to sender
  - ACK is used for flow control, error control, and congestion control
- ACK number sent is the next sequence number expected
- Delayed ACK: TCP receiver normally delays transmission of an ACK (for about 200ms)
- ACKs are not delayed when packets are received out of sequence
  - Why?



Lost segment

Acknowledgments in TCP

- Receiver sends ACK to sender
  - ACK is used for flow control, error control, and congestion control
- ACK number sent is the next sequence number expected
- Delayed ACK: TCP receiver normally delays transmission of an ACK (for about 200ms)
  - Why?
- ACKs are not delayed when packets are received out of sequence
  - Why?



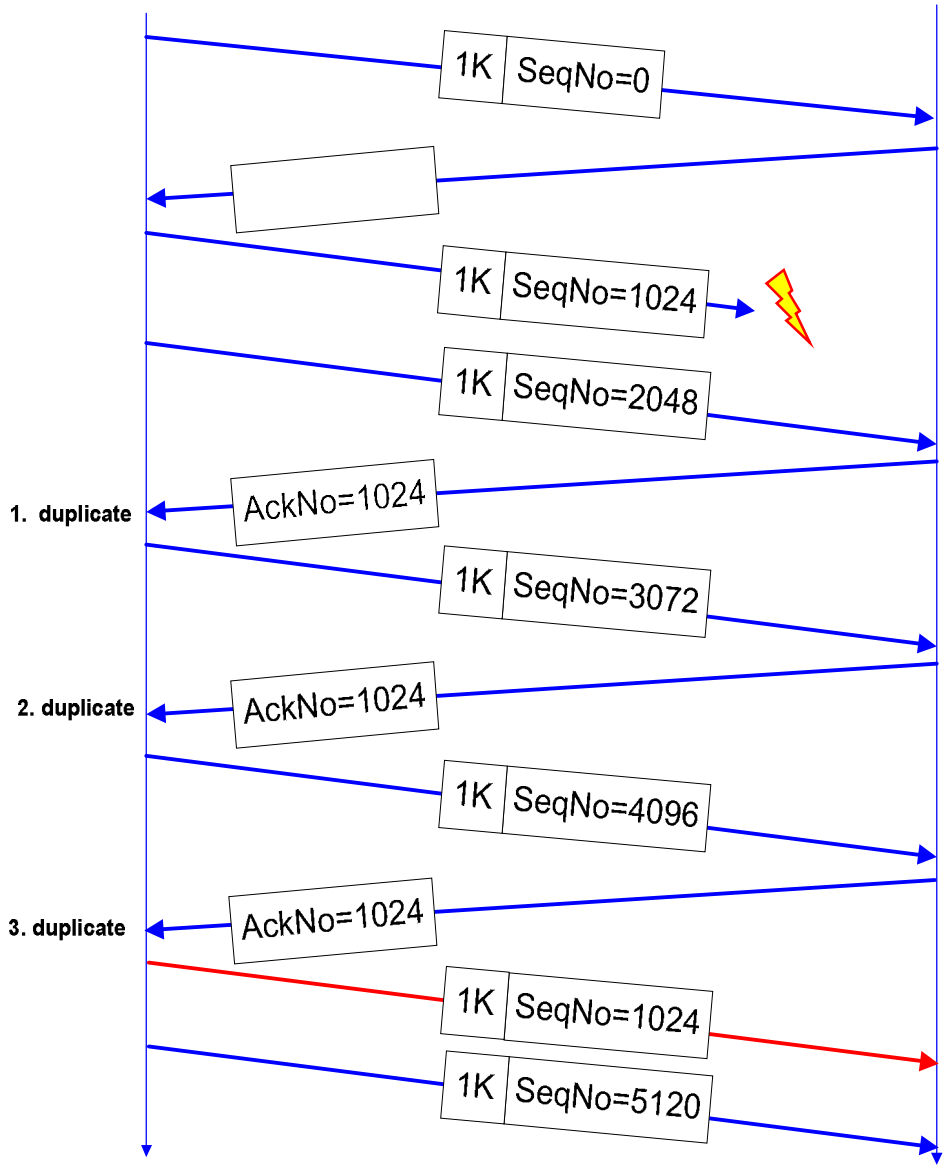
Out-of-order arrivals

Fast Retransmit

- If three or more duplicate ACKs are received in a row, the TCP sender believes that a segment has been lost.
- Then TCP performs a retransmission of what seems to be the missing segment, without waiting for a timeout to happen.
- Enter slow start:

  ssthresh = cwnd/2

  cwnd = 1

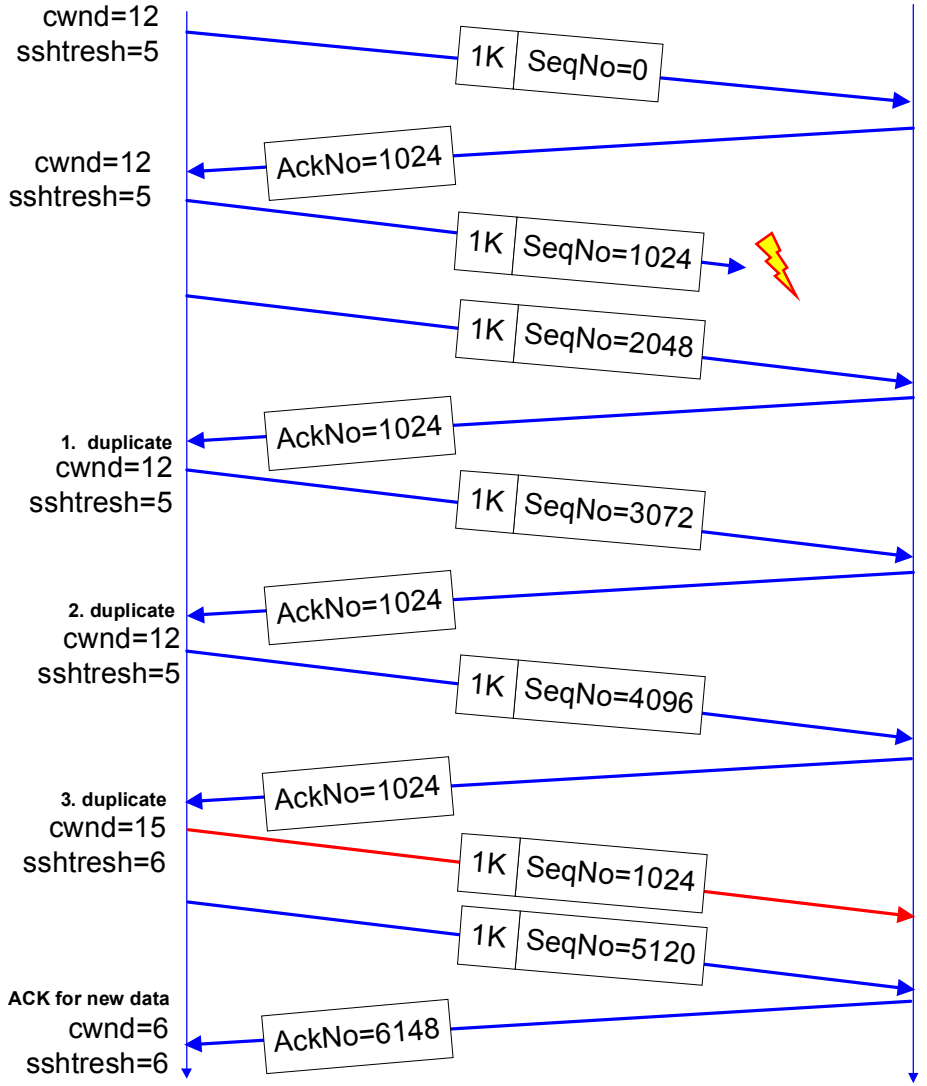


Fast Recovery

- Fast recovery avoids slow start after a fast retransmit
- **Intuition:** Duplicate ACKs indicate that data is getting through
- After three duplicate ACKs set:
  - Retransmit packet that is presumed lost
  - ssthresh = cwnd/2
  - cwnd = cwnd+3
  - (note the order of operations)
  - Increment cwnd by one for each additional duplicate ACK
- When ACK arrives that acknowledges "new data" (here: AckNo=6148), set:

  cwnd=ssthresh

  enter congestion avoidance



TCP Reno

- Duplicate ACKs:
  - Fast retransmit
  - Fast recovery

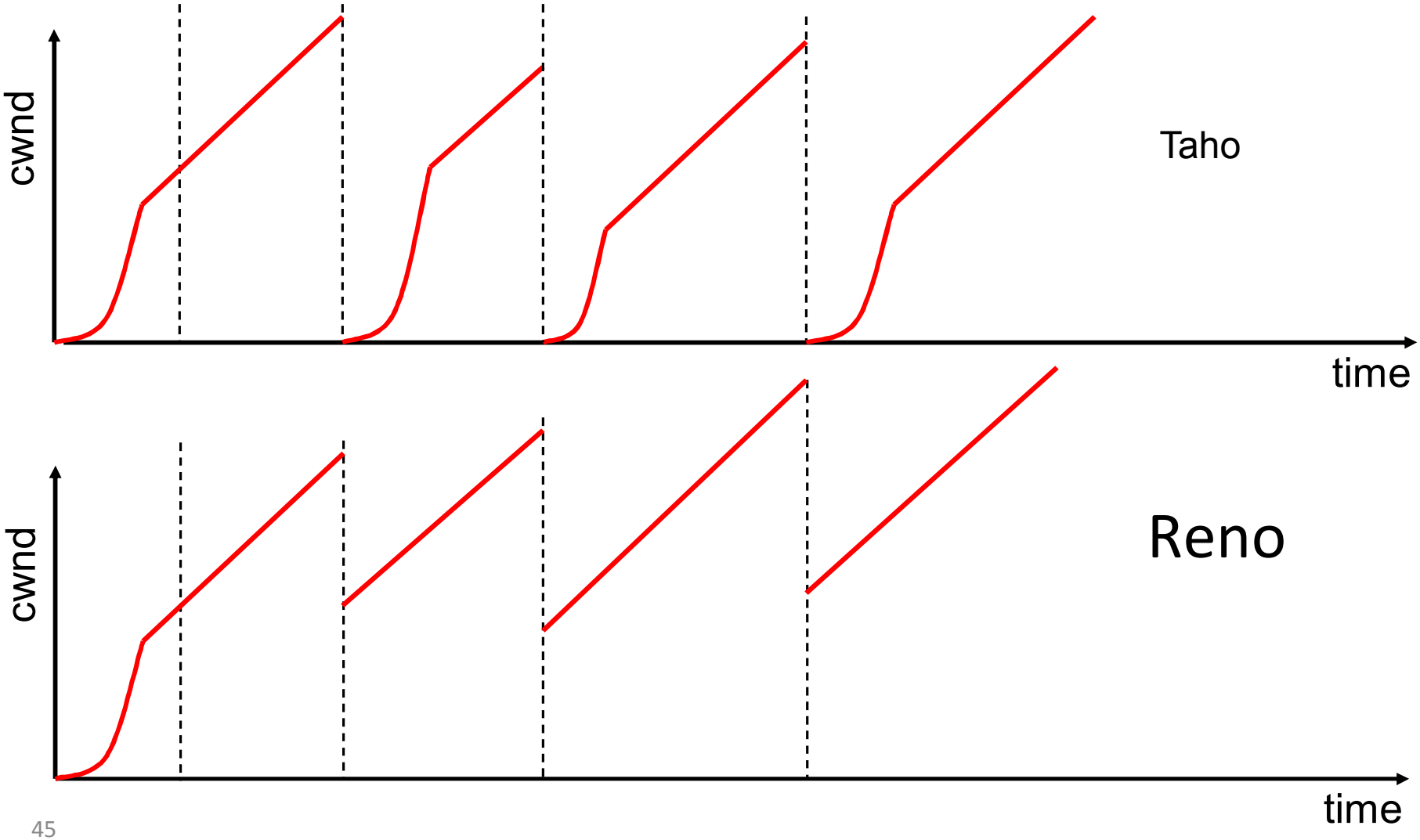
  → Fast Recovery avoids slow start

- Timeout:
  - Retransmit
  - Slow Start

- TCP Reno improves upon TCP Tahoe when a single packet is dropped in a round-trip time.

TCP Tahoe and TCP Reno

(for single segment losses)



TCP New Reno

- When multiple packets are dropped, Reno has problems
- Partial ACK:
  - Occurs when multiple packets are lost
  - A partial ACK acknowledges some, but not all packets that are outstanding at the start of a fast recovery, takes sender out of fast recovery

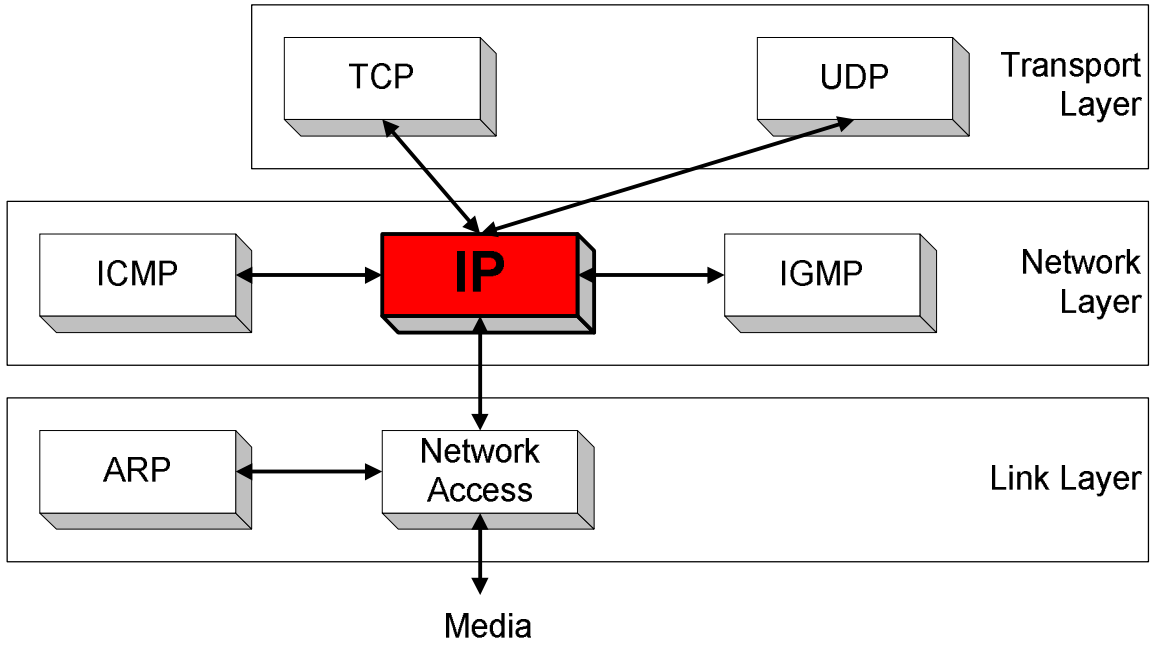
  →Sender has to wait until timeout occurs
- **New Reno:**
  - Partial ACK does not take sender out of fast recovery
  - Partial ACK causes retransmission of the segment following the acknowledged segment
- New Reno can deal with multiple lost segments without going to slow start

SACK

- SACK = Selective acknowledgment
- Issue: Reno and New Reno retransmit at most 1 lost packet per round trip time
- **Selective acknowledgments:** The receiver can acknowledge non-continuous blocks of data (SACK 0-1023, 1024-2047)
- Multiple blocks can be sent in a single segment.
- TCP SACK:
  - Enters fast recovery upon 3 duplicate ACKs
  - Sender keeps track of SACKs and infers if segments are lost. Sender retransmits the next segment from the list of segments that are deemed lost.

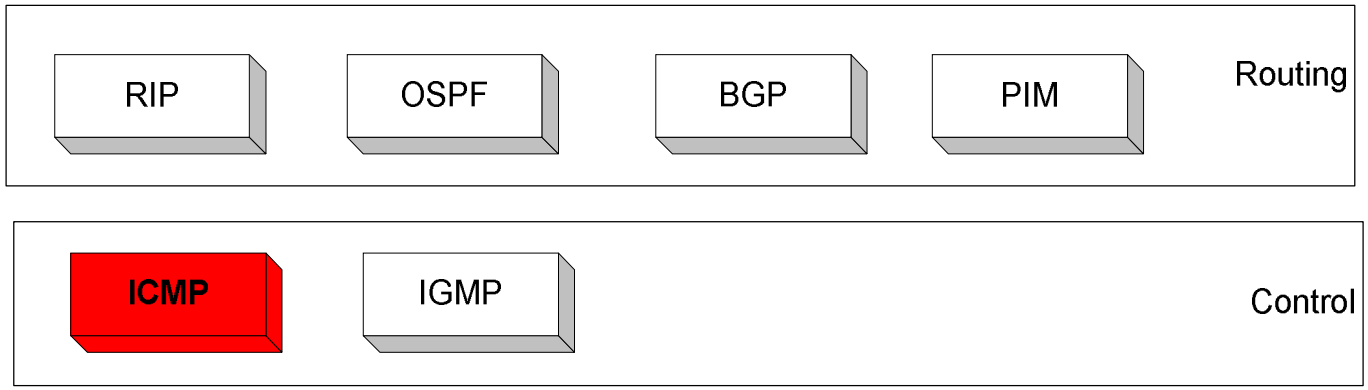
Back to IP - routing - level

- What is ICMP, how does routing work etc



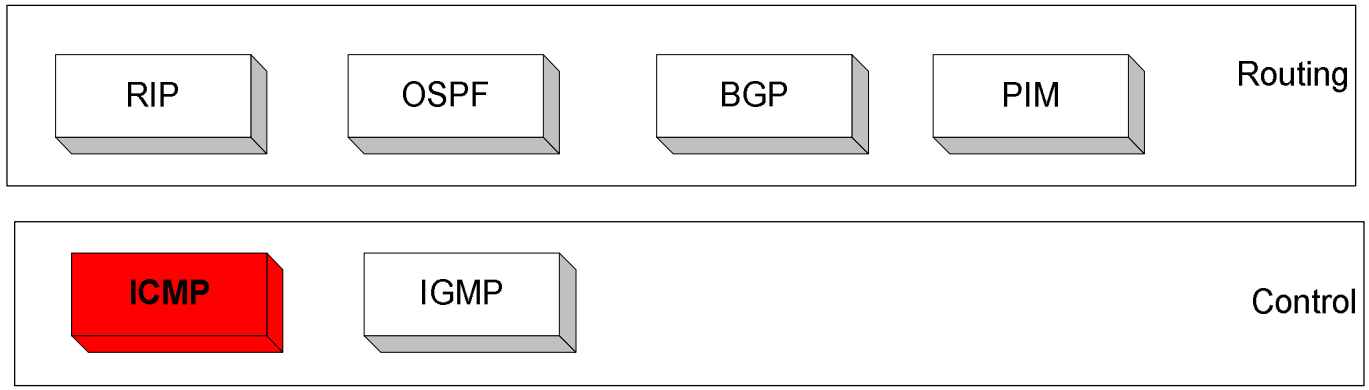
Overview

- The IP (Internet Protocol) relies on several other protocols to perform necessary control and routing functions:
  - Control functions (ICMP)
  - Multicast signaling (IGMP)
  - Setting up routing tables (RIP, OSPF, BGP, PIM, …)



Two levels

- RIP, OSPF etc for filling and **updating routing tables**, not actual routing of packages
- IP for **actual routing**, ICMP for **error messages and simple queries**

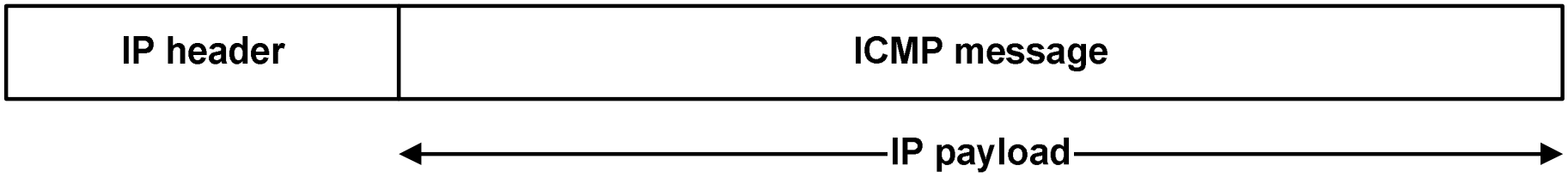


ICMP

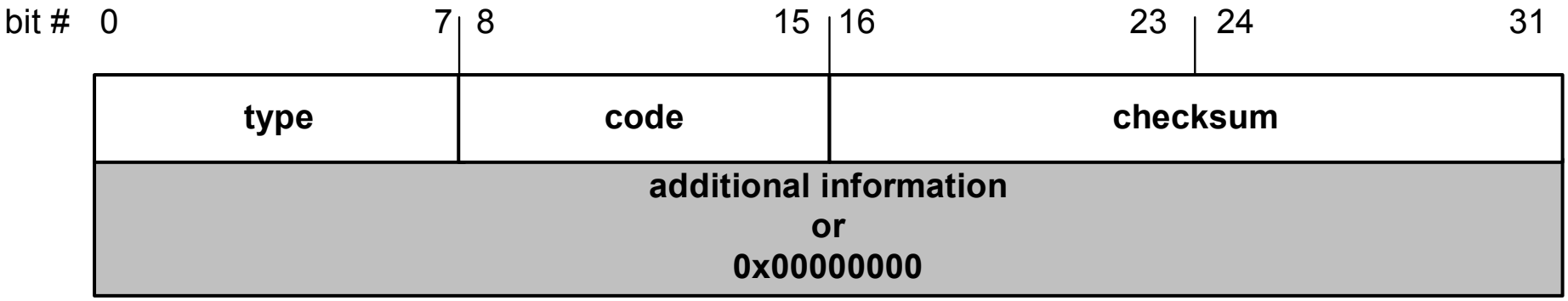
Overview

- The **Internet Control Message Protocol (ICMP)** is a helper protocol that supports IP with facility for
  - Error reporting
  - Simple queries

- ICMP messages are encapsulated as IP datagrams:



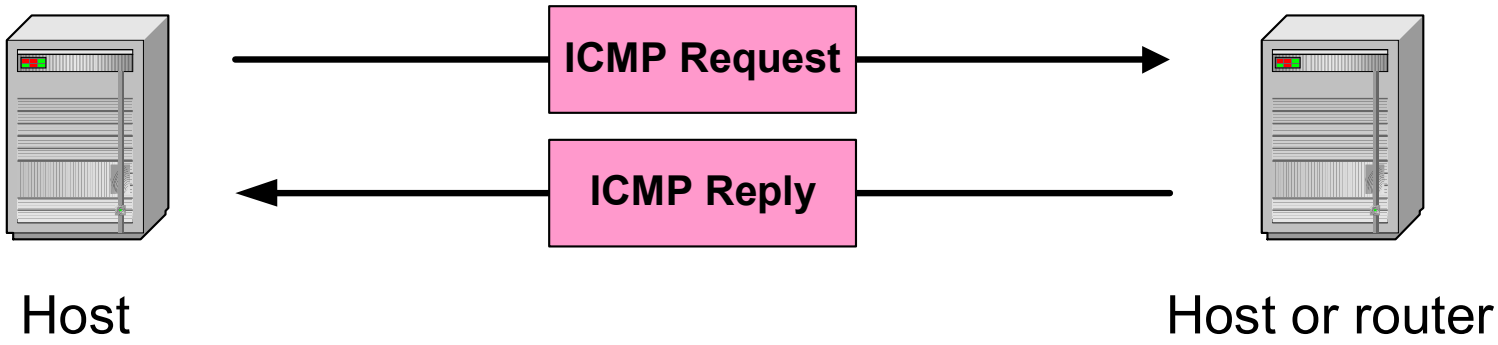
ICMP message format



- Type (1 byte): type of ICMP message
- Code (1 byte): subtype of ICMP message
- Checksum (2 bytes): similar to IP header checksum. Checksum is calculated over entire ICMP message

If there is no additional data, there are 4 bytes set to zero.
→ each ICMP messages is at least 8 bytes

ICMP Query message



- Request sent by host to a router or host
- Reply sent back to querying host

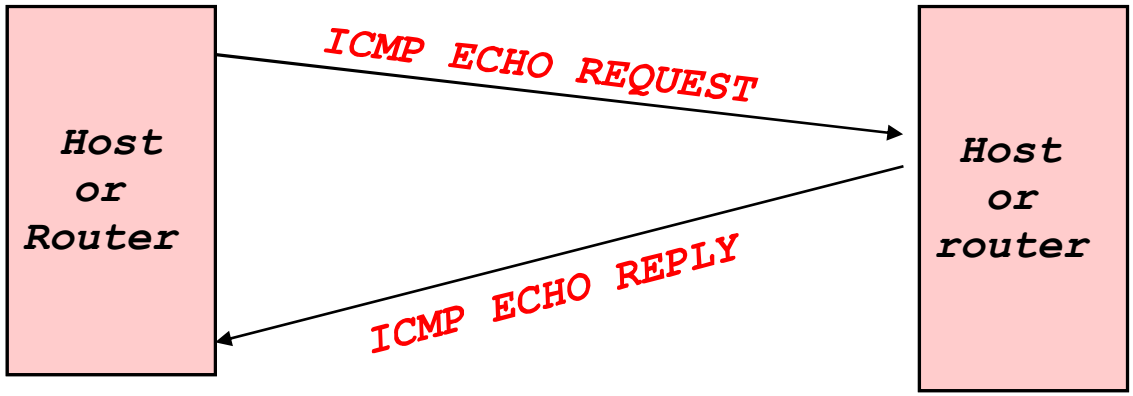
Example of ICMP Queries

Type/Code:	Description
8/0	Echo Request
0/0	Echo Reply
13/0	Timestamp Request
14/0	Timestamp Reply
10/0	Router Solicitation
9/0	Router Advertisement

The ping command uses Echo Request/ Echo Reply

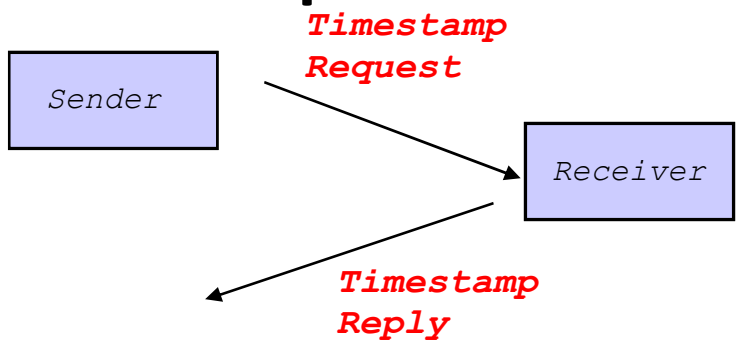
Example of a Query:
Echo Request and Reply

- Ping's are handled directly by the kernel
- Each Ping is translated into an ICMP Echo Request
- The Ping'ed host responds with an ICMP Echo Reply



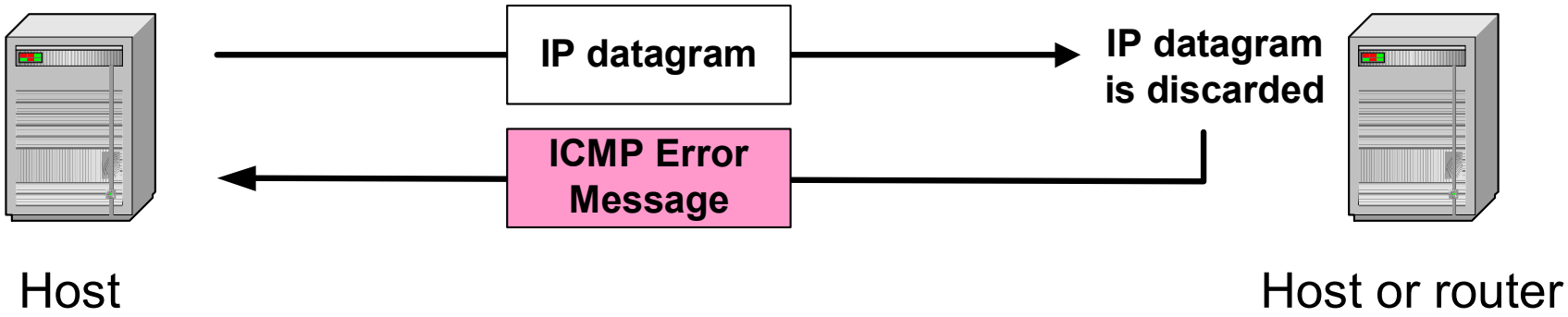
Example of a Query: ICMP Timestamp

- A system (host or router) asks another system for the current time.
- Time is measured in milliseconds after midnight UTC (Universal Coordinated Time) of the current day
- Sender sends a request, receiver responds with reply



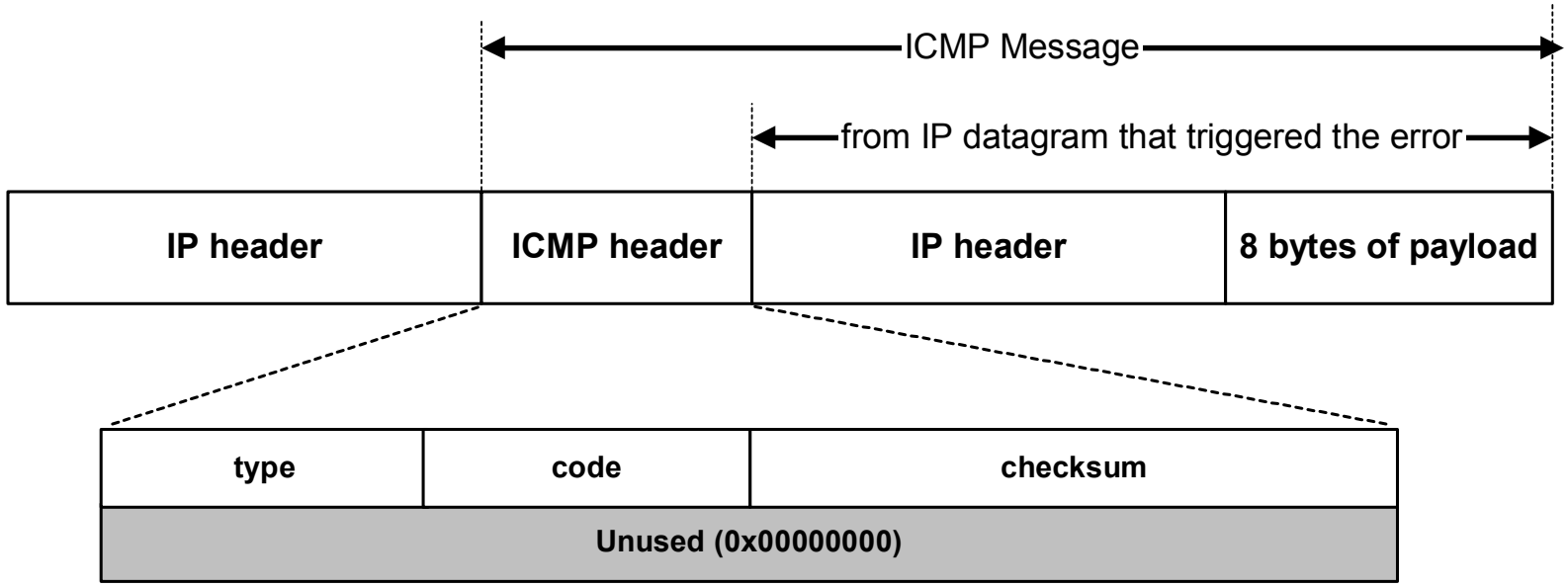
Type (= 17 or 18)	Code (=0)	Checksum
identifier		sequence number
32-bit sender timestamp		
32-bit receive timestamp		
32-bit transmit timestamp		

ICMP Error message



- **ICMP error messages report error conditions**
- **Typically sent when a datagram is discarded**
- **Error message is often passed from ICMP to the application program**

ICMP Error message



- ICMP error messages include the complete IP header and the first 8 bytes of the payload (typically: UDP, TCP)

Frequent ICMP Error message

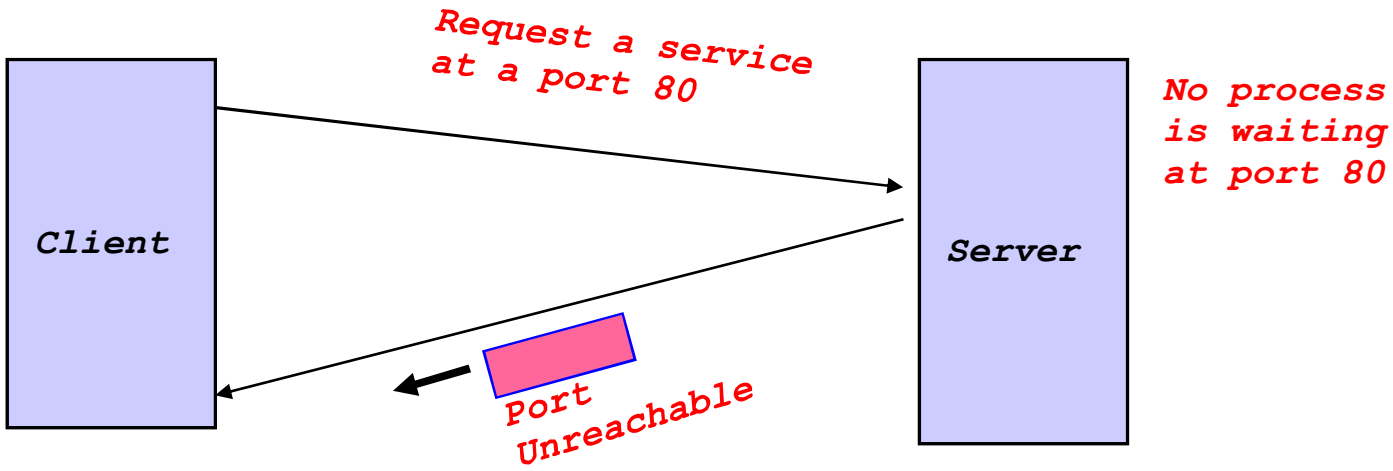
Type	Code	Description	
3	0–15	Destination unreachable	Notification that an IP datagram could not be forwarded and was dropped. The code field contains an explanation.
5	0–3	Redirect	Informs about an alternative route for the datagram and should result in a routing table update. The code field explains the reason for the route change.
11	0, 1	Time exceeded	Sent when the TTL field has reached zero (Code 0) or when there is a timeout for the reassembly of segments (Code 1)
12	0, 1	Parameter problem	Sent when the IP header is invalid (Code 0) or when an IP header option is missing (Code 1)

Some subtypes of the “Destination Unreachable”

Code	Description	Reason for Sending
0	Network Unreachable	No routing table entry is available for the destination network.
1	Host Unreachable	Destination host should be directly reachable, but does not respond to ARP Requests.
2	Protocol Unreachable	The protocol in the protocol field of the IP header is not supported at the destination.
3	Port Unreachable	The transport protocol at the destination host cannot pass the datagram to an application.
4	Fragmentation Needed and DF Bit Set	IP datagram must be fragmented, but the DF bit in the IP header is set.

Example: ICMP Port Unreachable

- RFC 792: If, in the destination host, the IP module cannot deliver the datagram because the indicated protocol module or process port is not active, the destination host may send a destination unreachable message to the source host.
- Scenario:



Some ICMP messages often blocked at firewalls

- Possible CISCO conf example

Access-list 101 (to be applied to the external interface)

```
access-list 101 deny icmp any any fragments
access-list 101 permit icmp any any echo-reply
access-list 101 permit icmp any any time-exceeded
access-list 101 permit icmp any any packet-too-big
access-list 101 deny icmp any any
```

Access-list 102 (to be applied to the internal interface)

```
access-list 102 deny icmp any any fragments
access-list 102 permit icmp any any echo-request
access-list 102 permit icmp any any time-exceeded
access-list 102 permit icmp any any packet-too-big
access-list 102 deny icmp any any
```

Some ICMP messages often blocked at firewalls

- Possible Linux IPTABLES conf example

```
iptables -A INPUT -p icmp --fragment -j DROP
iptables -A INPUT -p icmp --icmp-type echo-reply -j ACCEPT
iptables -A OUTPUT -p icmp --icmp-type echo-request -j ACCEPT
iptables -A INPUT -p icmp --icmp-type time-exceeded -j ACCEPT
iptables -A INPUT -p icmp --icmp-type fragmentation-needed -j ACCEPT
iptables -A OUTPUT -p icmp --icmp-type time-exceeded -j ACCEPT
iptables -A OUTPUT -p icmp --icmp-type fragmentation -needed -j ACCEPT
iptables -A INPUT -p ICMP -j DROP
iptables -A OUTPUT -p ICMP -j DROP
```

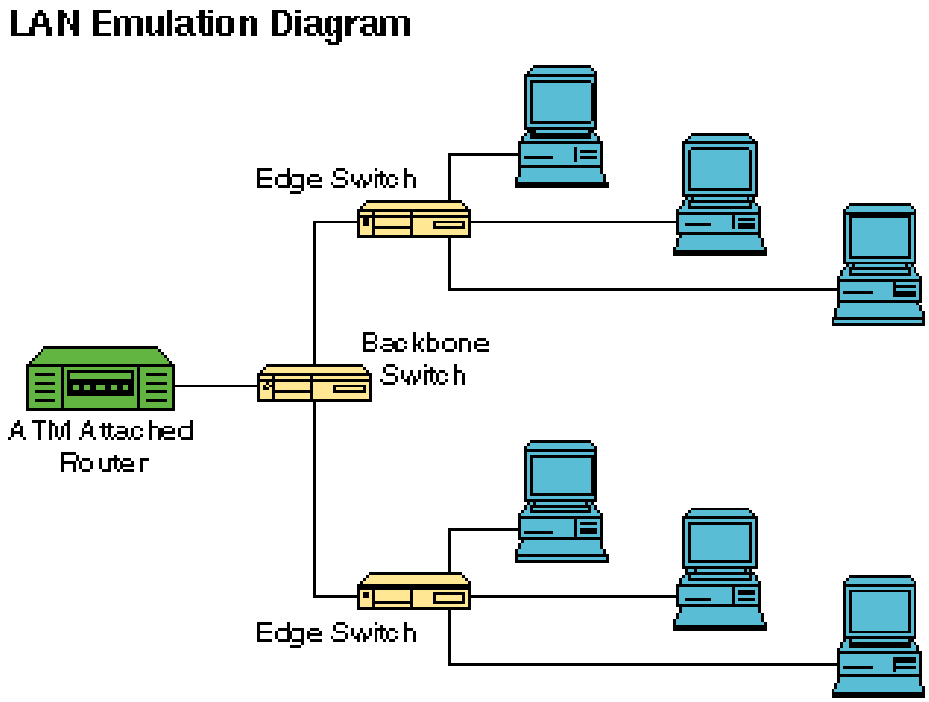
Routers

Switches and routers

- **Switch**: connects hosts on local ethernet
  - send unmodified frames to MAC address
- **Router**: connects separate ethernet networks
  - take out IP package from frame
  - check if IP must be sent to another local network
  - if yes:
    - modify IP header (TTL, checksum),
    - package into new frame
    - send to another network

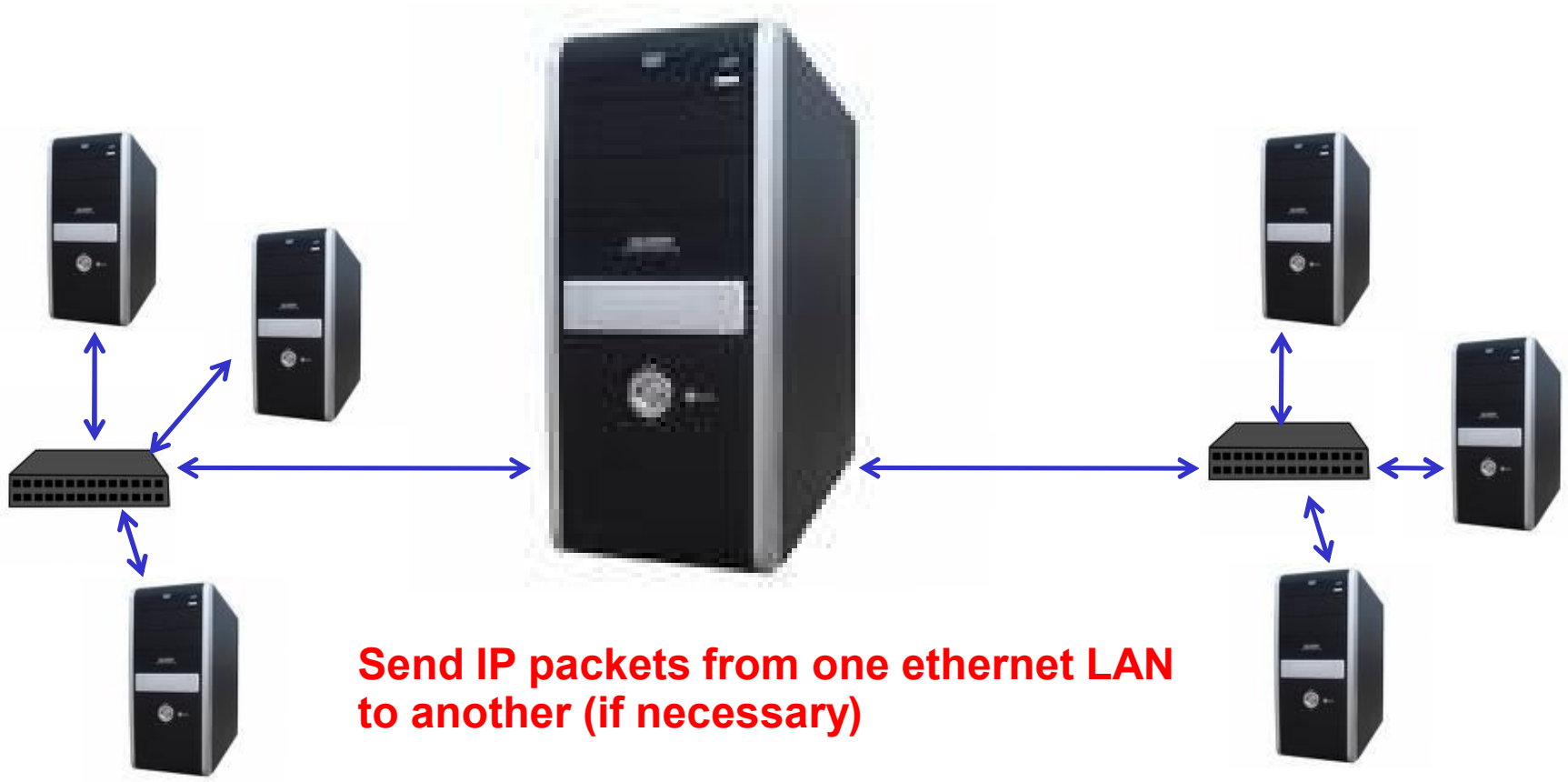
Switch, not router

- Ethernet case: send ethernet frames
  - to everybody on the net (hub)
  - or learned ethernet plugs for mac addresses (switch)

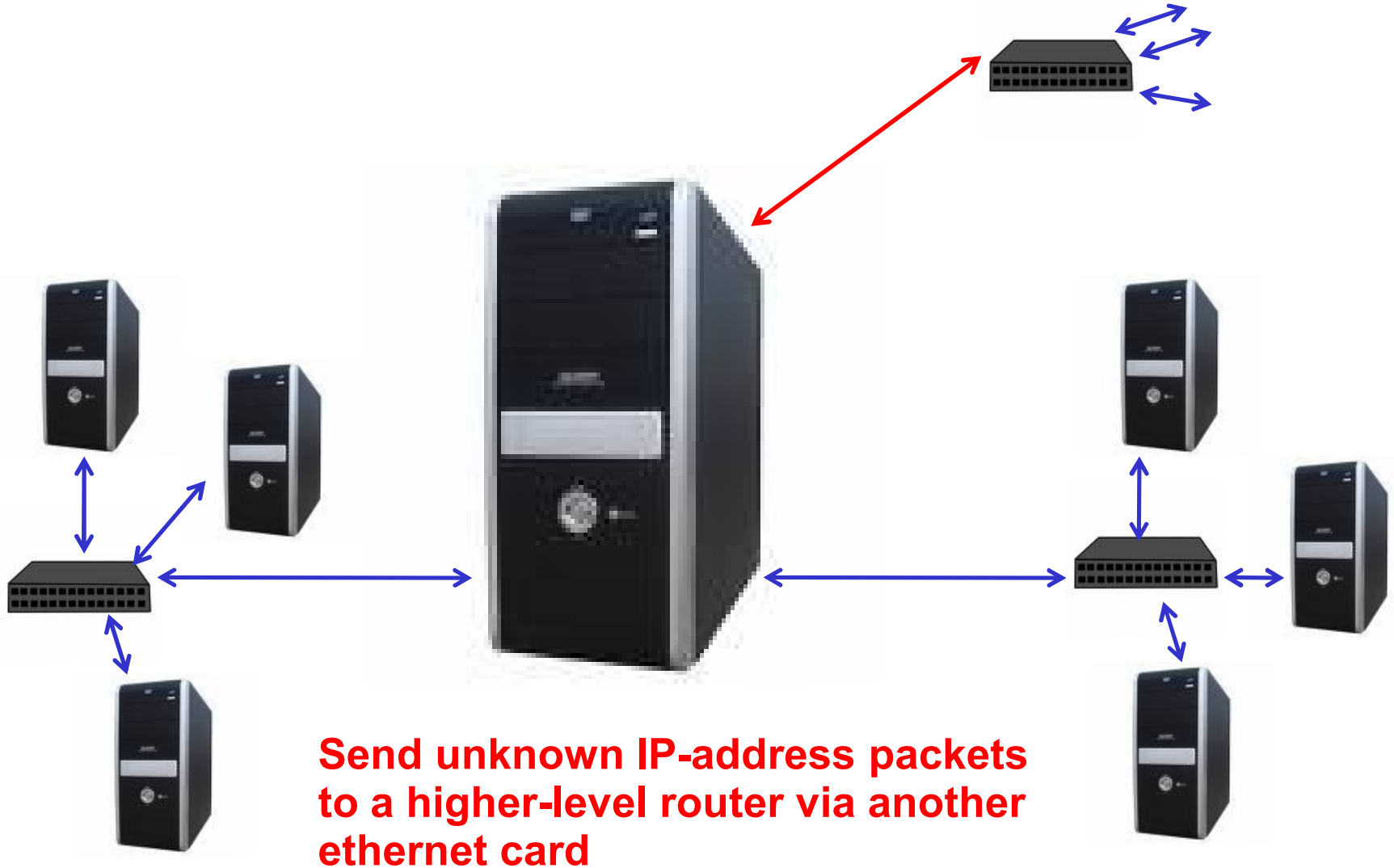


Simple router

- Ordinary PC with at least two ethernet cards



Another ordinary case …



Send unknown IP-address packets to a higher-level router via another ethernet card

What special stuff router must contain

- **Routing table**:
  - Which IP addresses should be sent to
    - Ethernet card 1
    - Ethernet card 2
    - ….
- **Simple IP package** mangling and messaging software:
  - Increase TTL field, compute new checksum
  - Send ICMP error messages as necessary
  - Check routing table for each IP package

Large specialised routers

A lot of ethernet plugs and very fast operation

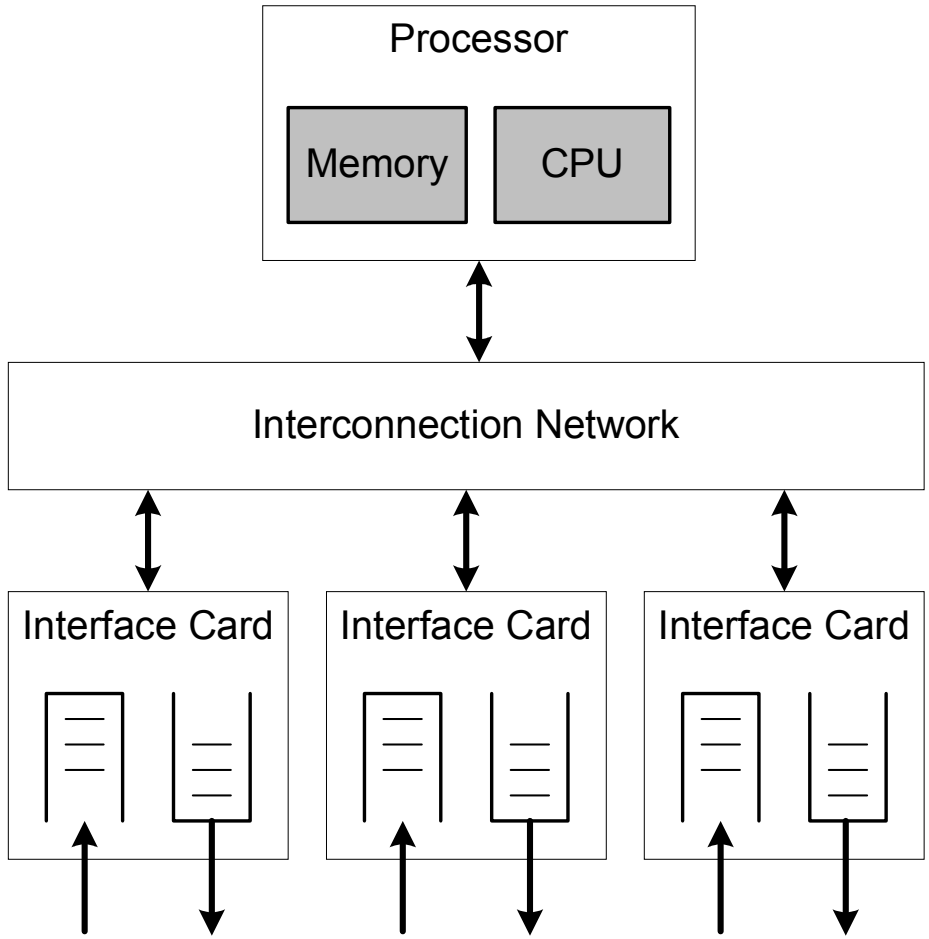
Layer 1, 2, ... "switches"

Layer nr means what OSI layer header data is used when routing

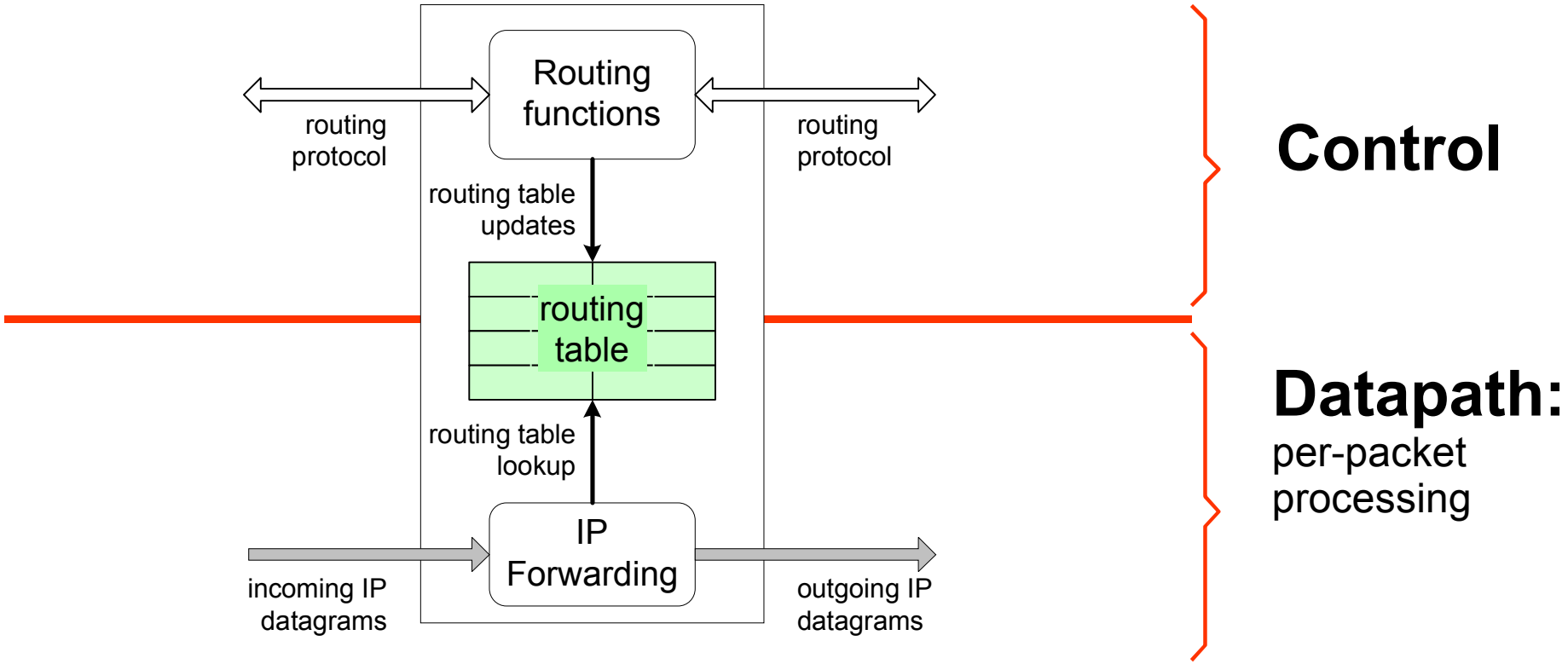
- Layer 1: ethernet hub: just connection
- Layer 2: ethernet switch: learn MAC/plug pairs
- **Layer 3: IP-level router**
- Layer 4-7: Look also into internal TCP etc headers

Router Components

- Hardware components of a router:
  - Network interfaces
  - Interconnection network
  - Processor with a memory and CPU
- **PC router:**
  - interconnection network is the (PCI) bus and interface cards are NICs
  - All forwarding and routing is done on central processor
- **Commercial routers:**
  - Interconnection network and interface cards are sophisticated
  - Processor is only responsible for control functions **(route processor)**
  - Almost all forwarding is done on interface cards



Functional Components



Routing and Forwarding

Routing functions include:

- route calculation
- maintenance of the routing table
- execution of routing protocols

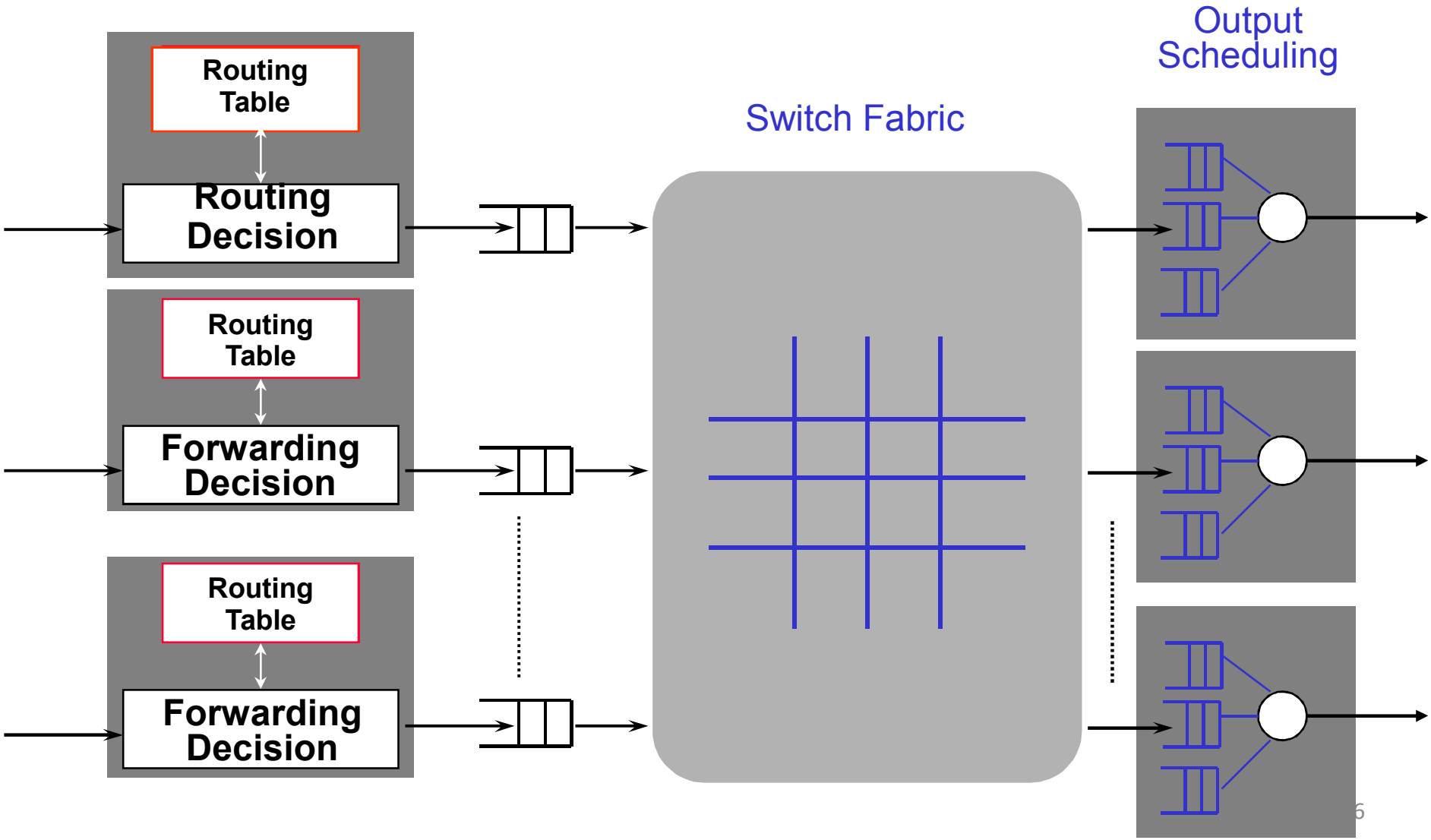
- On commercial routers handled by a single general purpose processor, called *route processor*

IP forwarding is per-packet processing

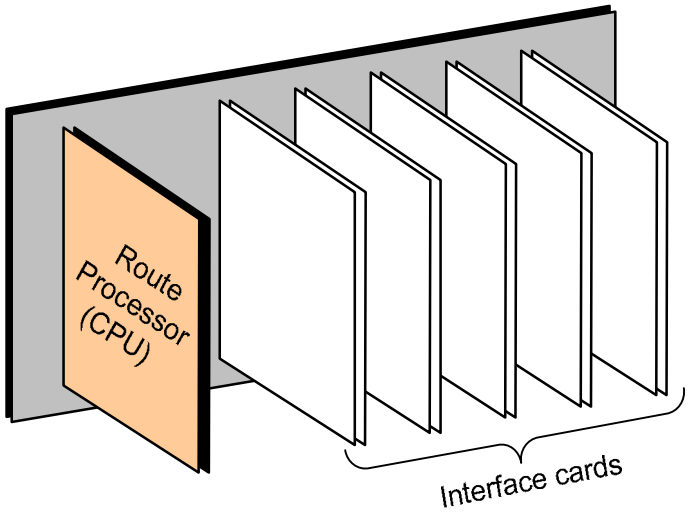
- On high-end commercial routers, IP forwarding is distributed
- Most work is done on the interface cards

Basic Architectural Components

Per-packet processing



Slotted Chassis



- Large routers are built as a slotted chassis
  - Interface cards are inserted in the slots
  - Route processor is also inserted as a slot

- This simplifies repairs and upgrades of components

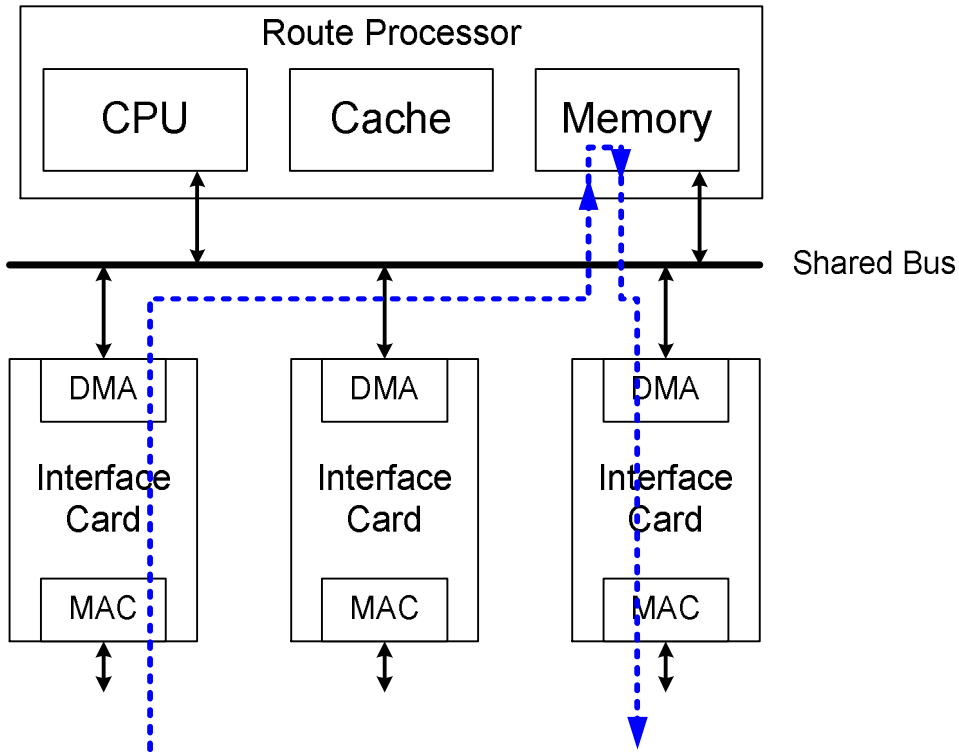
Evolution of Router Architectures

- Early routers were essentially general purpose computers
- Today, high-performance routers resemble supercomputers
    - Exploit parallelism
    - Special hardware components

- Until 1980s (1st generation): standard computer
- Early 1990s (2nd generation): delegate to interfaces
- Late 1990s (3rd generation): Distributed architecture
- Today: Distributed over multiple racks

1st Generation Routers

- This architecture is still used in low end routers
- Arriving packets are copied to main memory via direct memory access (DMA)
- Interconnection network is a backplane (shared bus)
- All IP forwarding functions are performed in the central processor.
- Routing cache at processor can accelerate the routing table lookup.
- Drawbacks:
  - Forwarding Performance is limited by CPU
  - Capacity of shared bus limits the number of interface cards that can be connected



2nd Generation Routers

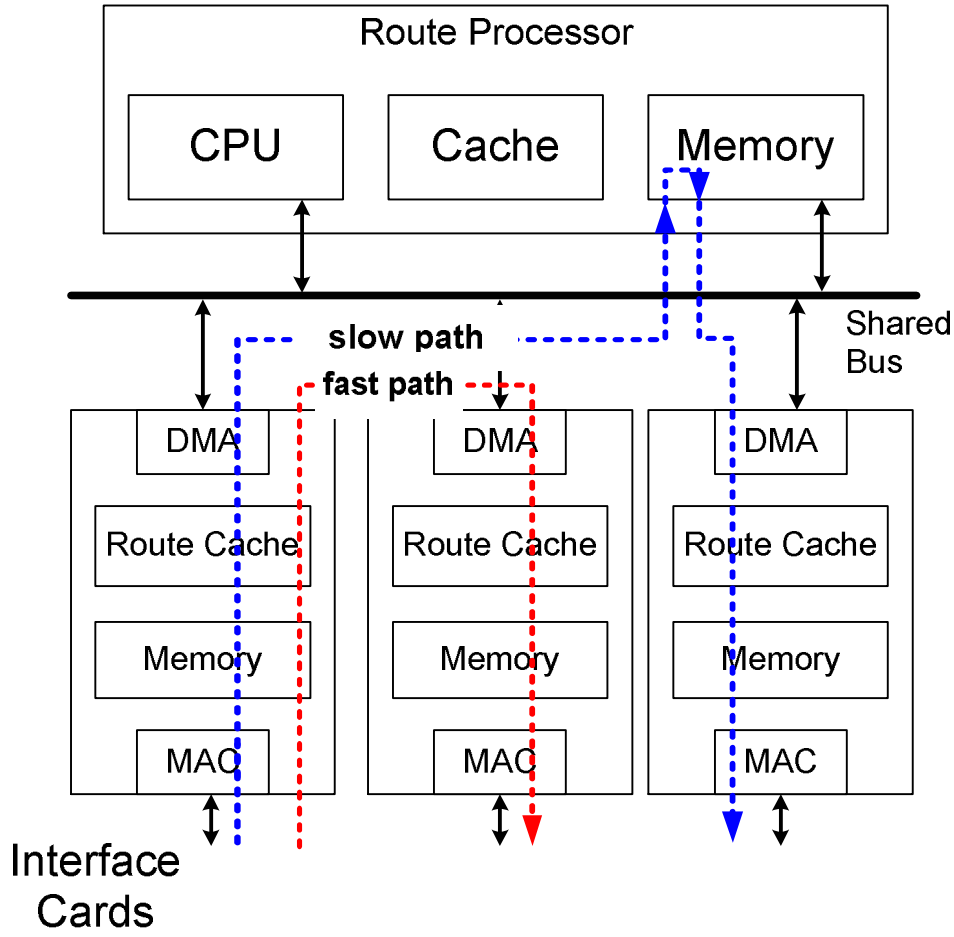
- Keeps shared bus architecture,

  but offloads most IP forwarding to interface cards
- Interface cards have local route cache and processing elements

Fast path: If routing entry is found in local cache, forward packet directly to outgoing interface

Slow path: If routing table entry is not in cache, packet must be handled by central CPU

- Drawbacks: Shared bus is still bottleneck

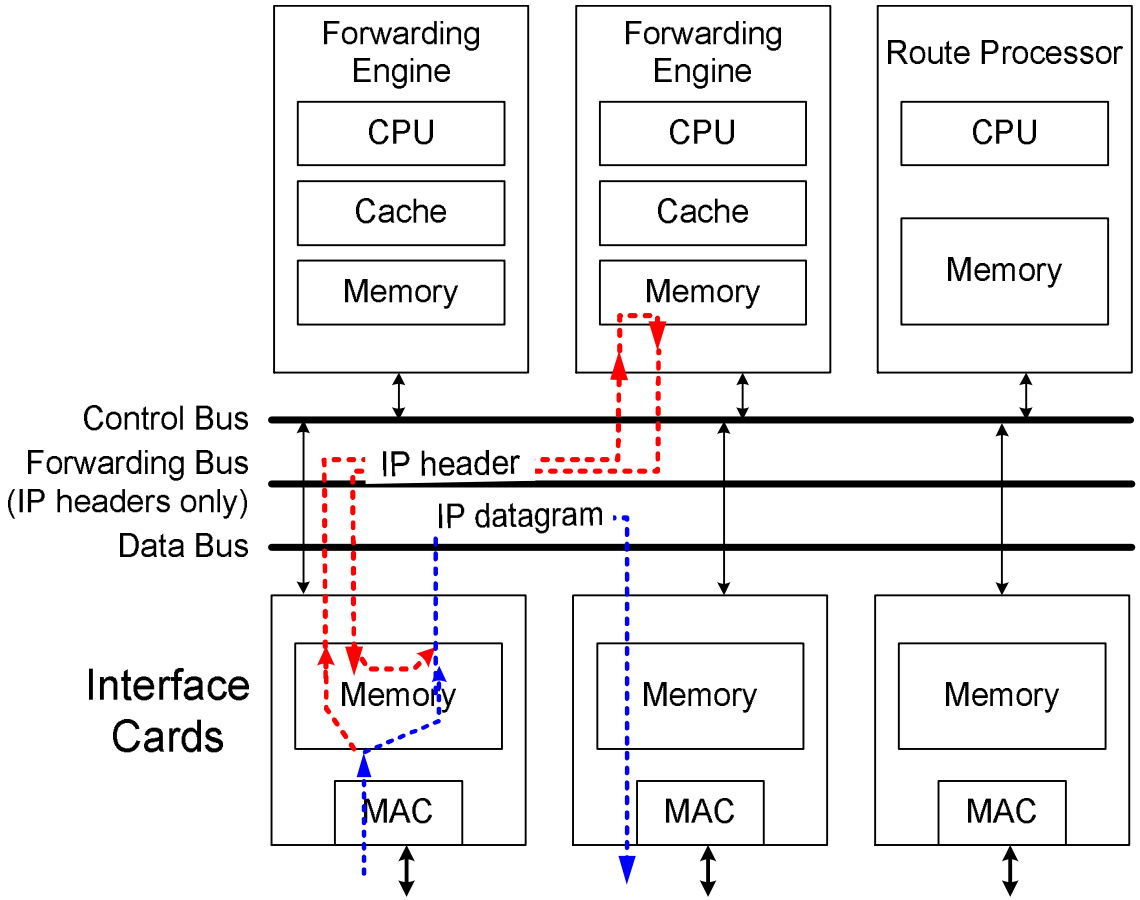


Another 2nd Generation Architecture

- IP forwarding is done by separate components (Forwarding Engines)

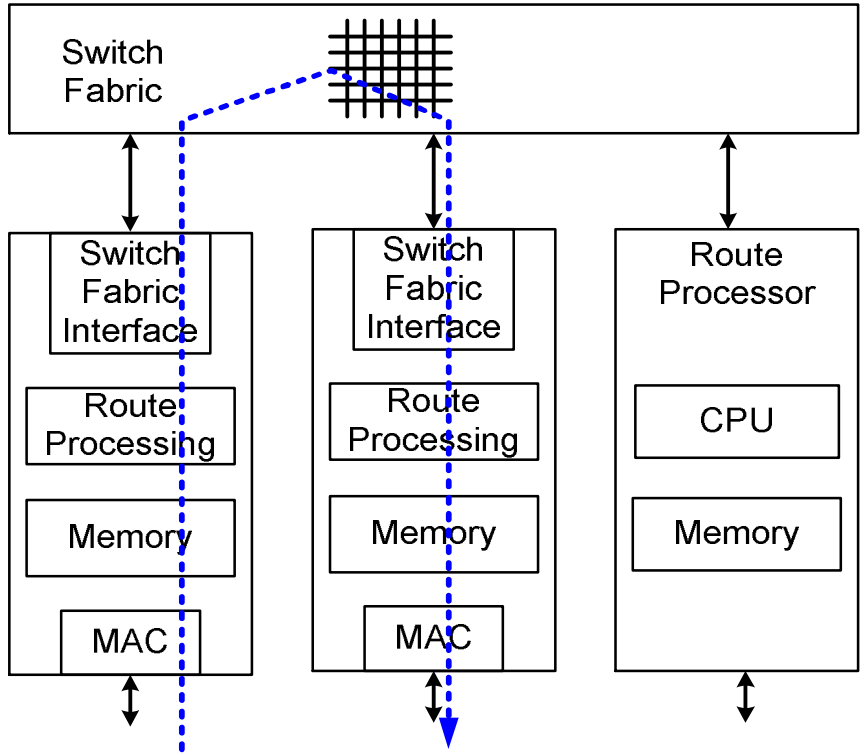
Forwarding operations:

1. Packet received on interface: Store the packet in local memory. Extracts IP header and sent to one forwarding engine
2. Forwarding engine does lookup, updates IP header, and sends it back to incoming interface
3. Packet is reconstructed and sent to outgoing interface.



3rd Generation Architecture

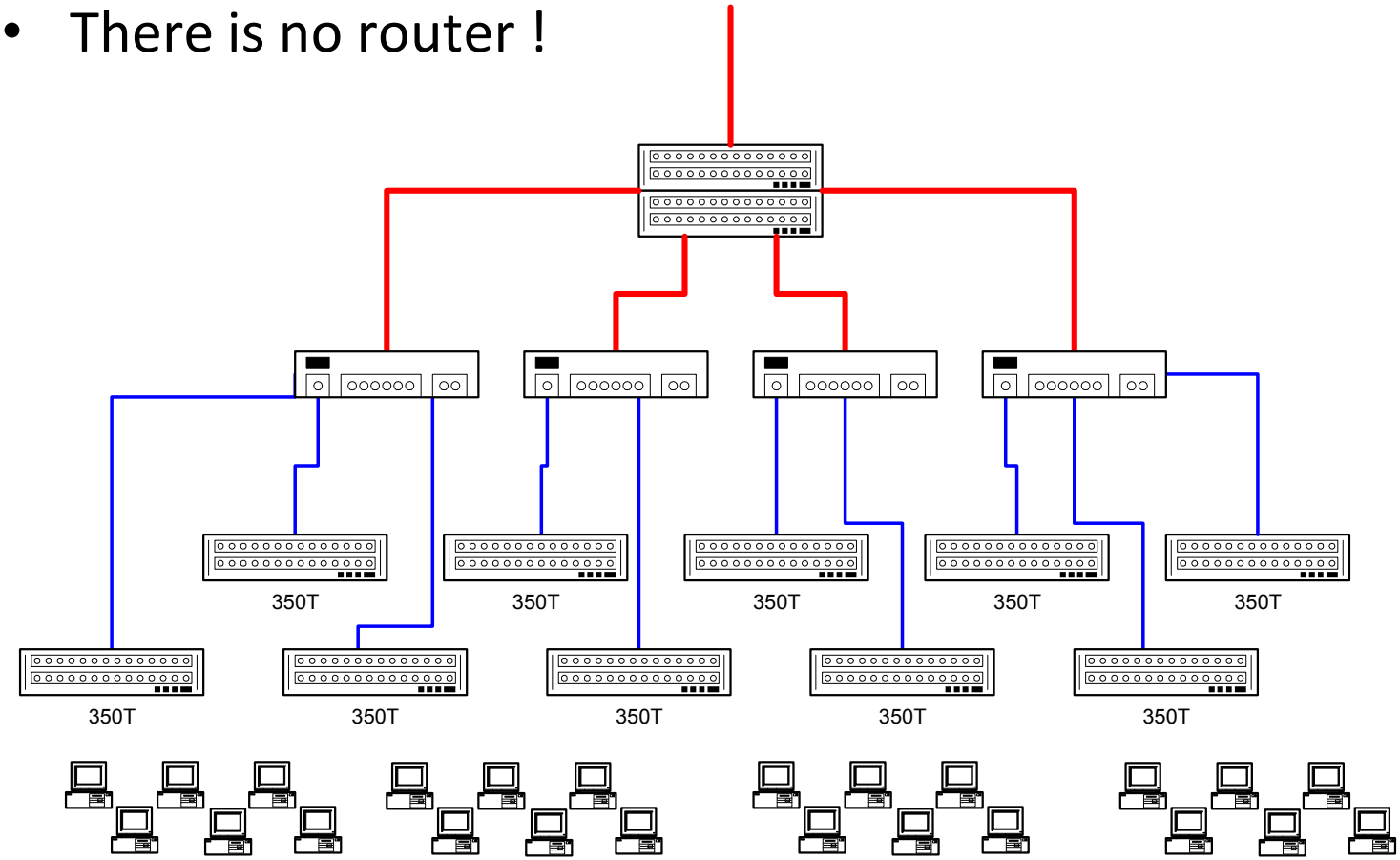
- Interconnection network is a switch fabric (e.g., a crossbar switch)
- **Distributed architecture:**
  - Interface cards operate independent of each other
  - No centralized processing for IP forwarding
- These routers can be scaled to many hundred interface cards and to aggregate capacity of > 1 Terabit per second



Network topology examples

Routing Example: Univ. of Virginia CS Department Network

- Design of the network architecture (Spring 2000)
- There is no router !



Problems with last slide ...

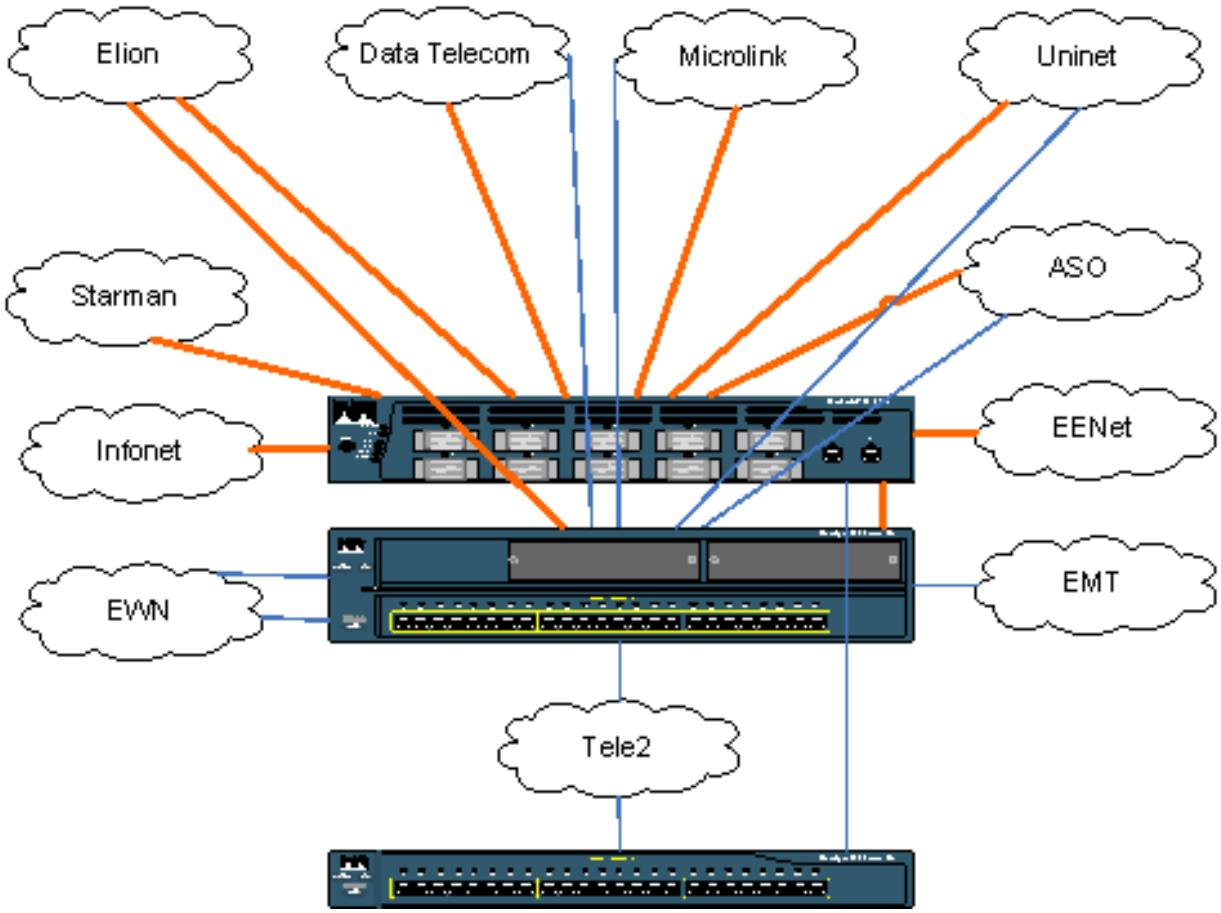
- In case no routers are used, multicast travels to all hosts on the network
- Lots of services are multicast:
  - arp
  - dhcp
  - several windows services
  - ...
- Multicasts would overwhelm the network

Routing example: TTU

- Information from Andres Lepp (lõvi)
- See a separate pdf with an overview schema
- Local switches/switch hierarchies are connected with ca 5 routers (level 3 switches)
- Everything on the wire is ethernet
- External link is a gigabit fibre ethernet to EENET router

Routing examples: Estonia

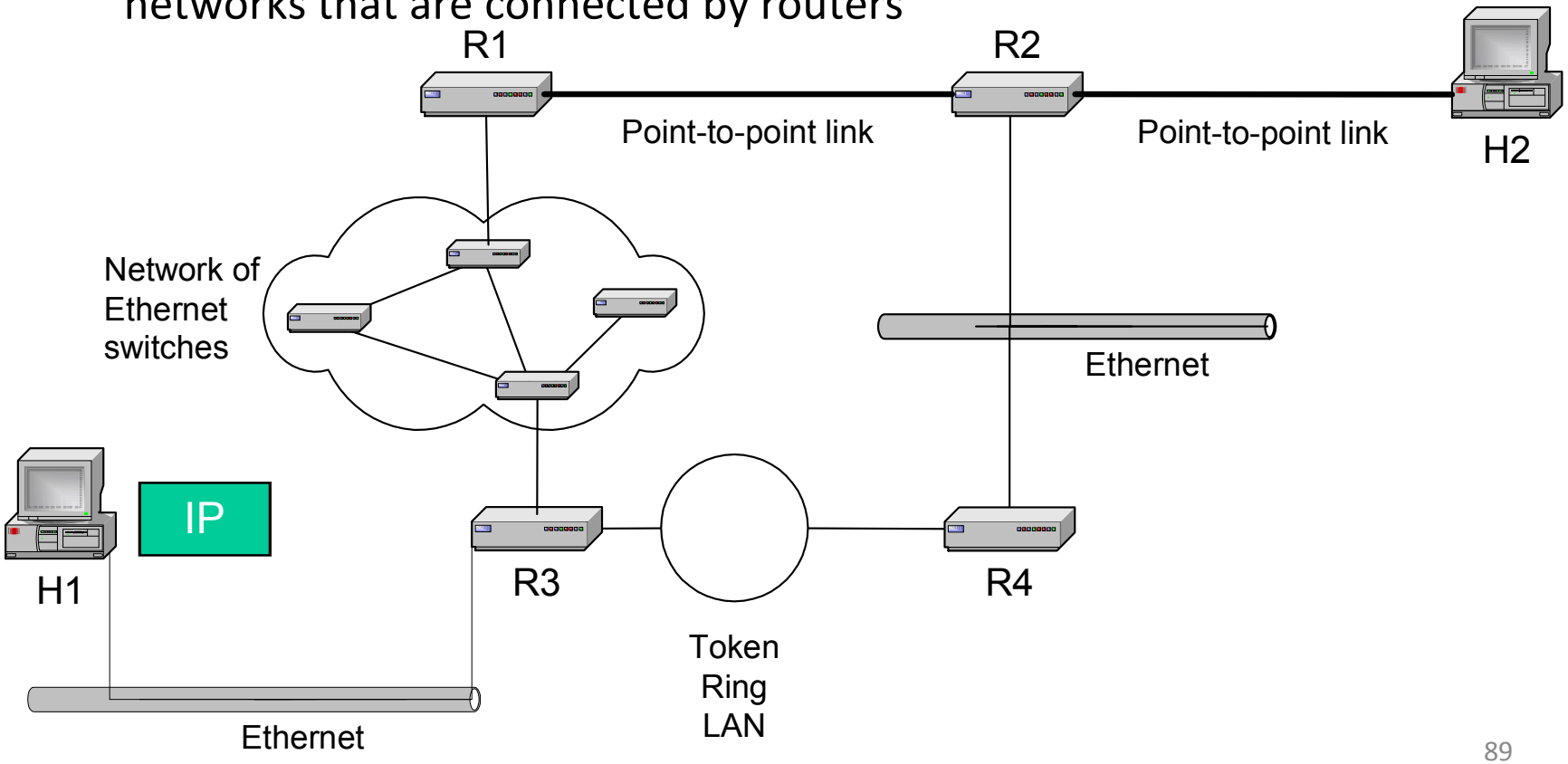
- TIX-LAN (http://tix.estpak.ee/)



Routing: IP forwarding

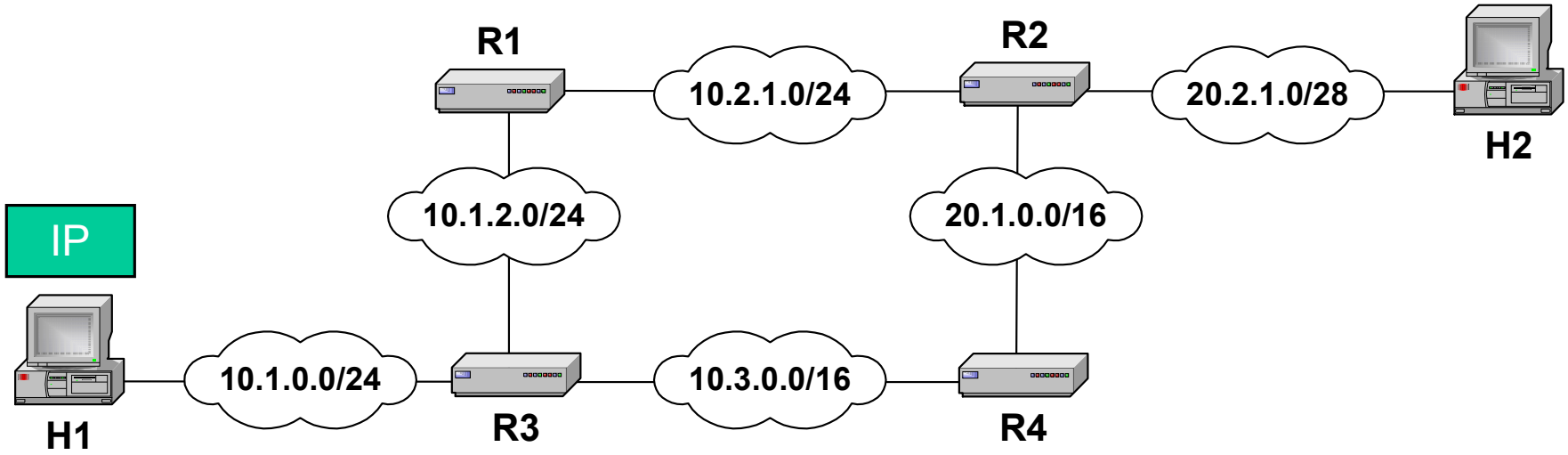
Delivery of an IP datagram

- View at the data link layer layer:
  - Internetwork is a collection of LANs or point-to-point links or switched networks that are connected by routers



Delivery of an IP datagram

- View at the IP layer:
  - An IP network is a logical entity with a network number
  - We represent an IP network as a “cloud”
  - The IP delivery service takes the view of clouds, and ignores the data link layer view



Tenets of end-to-end delivery of datagrams

The following conditions must hold so that an IP datagram can be successfully delivered

1. The network prefix of an IP destination address must correspond to a unique data link layer network (=LAN or point-to-point link or switched network).
   (The reverse need not be true!)
2. Routers and hosts that have a common network prefix must be able to exchange IP dagrams using a data link protocol (e.g., Ethernet, PPP)
3. Every data link layer network must be connected to at least one other data link layer network via a router.

Routing tables

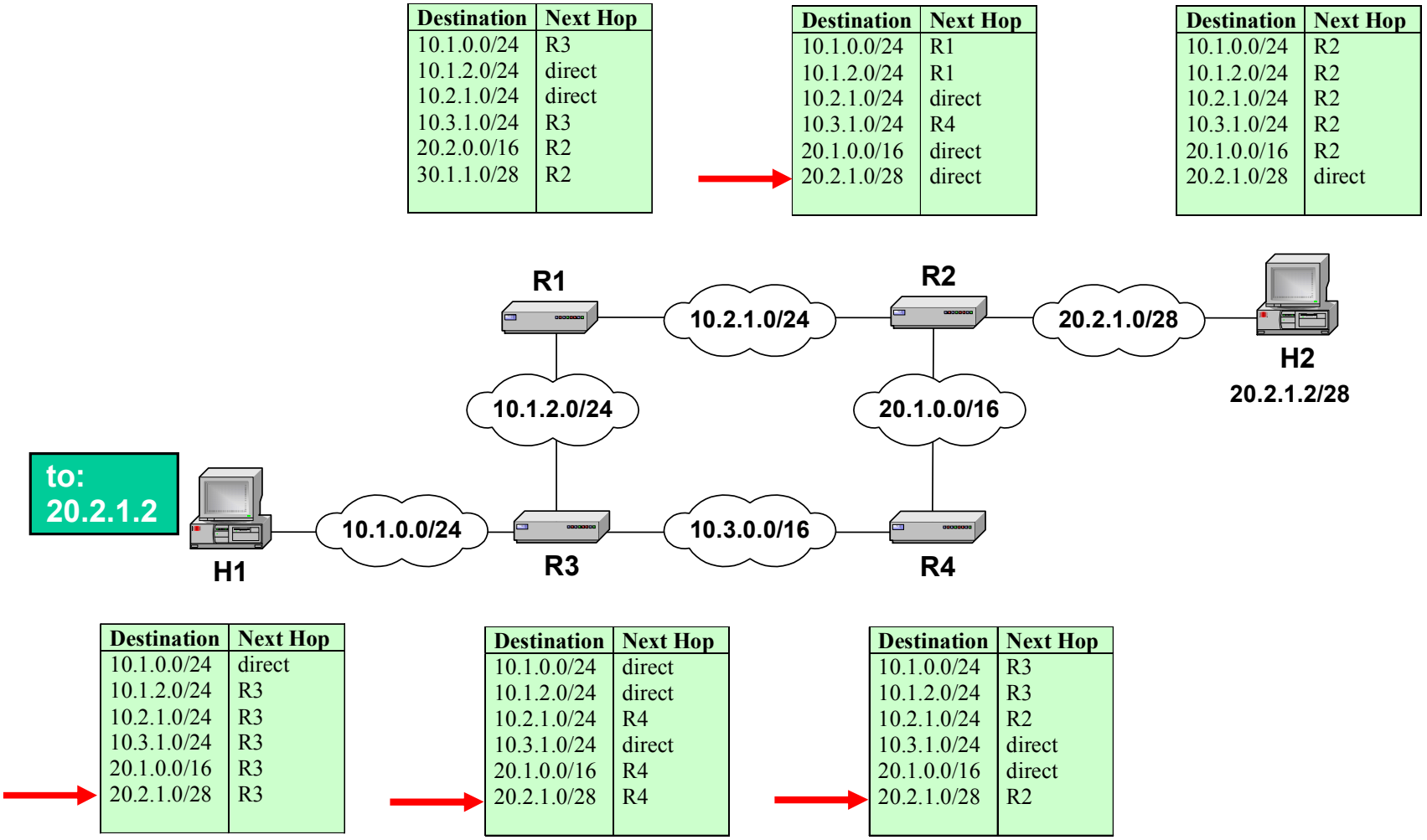
- Each router and each host keeps a **routing table** which tells the router how to process an outgoing packet
- Main columns:
  1. **Destination address:** where is the IP datagram going to?
  2. **Next hop:** how to send the IP datagram?
  3. **Interface:** what is the output port?
- Next hop and interface column can often be summarized as one column
- Routing tables are set so that datagrams gets closer to the its destination

Routing table of a host or router

IP datagrams can be directly delivered ("direct") or is sent to a router ("R4")

Destination	Next Hop	interface
10.1.0.0/24	direct	eth0
10.1.2.0/24	direct	eth0
10.2.1.0/24	R4	serial0
10.3.1.0/24	direct	eth1
20.1.0.0/16	R4	eth0
20.2.1.0/28	R4	eth0

Delivery with routing tables

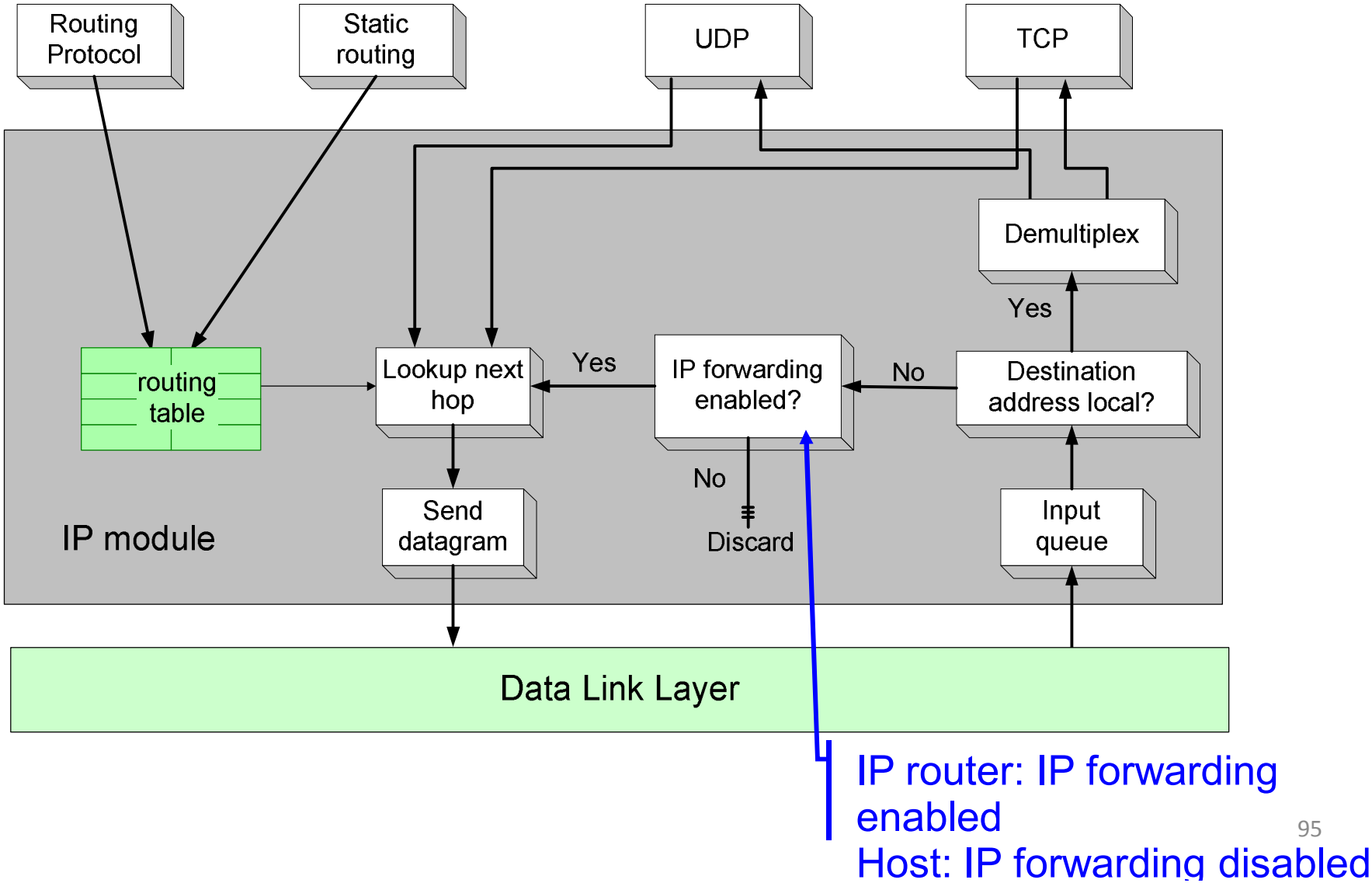


Delivery of IP datagrams

- There are two distinct processes to delivering IP datagrams:
  1. **Forwarding:** How to pass a packet from an input interface to the output interface?
  2. **Routing:** How to find and setup the routing tables?

- Forwarding must be done as fast as possible:
  - on routers, is often done with support of hardware
  - on PCs, is done in kernel of the operating system
- Routing is less time-critical
  - On a PC, routing is done as a background process

Processing of an IP datagram in IP



Processing of an IP datagram in IP

- Processing of IP datagrams is very similar on an IP router and a host
- **Main difference:**
  "IP forwarding" is enabled on router and disabled on host

- **IP forwarding enabled**
  → if a datagram is received, but it is not for the local system, the datagram will be sent to a different system
- **IP forwarding disabled**
  → if a datagram is received, but it is not for the local system, the datagram will be dropped

Processing of an IP datagram at a router

1. IP header validation
2. Process options in IP header
3. Parsing the destination IP address
4. Routing table lookup
5. Decrement TTL
6. Perform fragmentation (if necessary)
7. Calculate checksum
8. Transmit to next hop
9. Send ICMP packet (if necessary)

Routing table lookup

- When a router or host need to transmit an IP datagram, it performs a routing table lookup
- **Routing table lookup:** Use the IP destination address as a key to search the routing table.
- Result of the lookup is the IP address of a next hop router, and/or the name of a network interface

Destination address	**Next hop/ interface**
network prefix *or* host IP address *or* loopback address *or* default route	IP address of next hop router *or* Name of a network interface

Type of routing table entries

- **Network route**
  - Destination addresses is a network address (e.g., 10.0.2.0/24)
  - Most entries are network routes
- **Host route**
  - Destination address is an interface address (e.g., 10.0.1.2/32)
  - Used to specify a separate route for certain hosts
- **Default route**
  - Used when no network or host route matches
  - The router that is listed as the next hop of the default route is the **default gateway (for Cisco: "gateway of last resort)**
- **Loopback address**
  - Routing table for the loopback address (127.0.0.1)
  - The next hop lists the loopback (lo0) interface as outgoing interface

Routing table lookup: Longest Prefix Match

- **Longest Prefix Match:** Search for the routing table entry that has the longest match with the prefix of the destination IP address

1. Search for a match on all 32 bits
2. Search for a match for 31 bits

…..

32. Search for a mach on 0 bits

Host route, loopback entry
→ 32-bit prefix match

Default route is represented as 0.0.0.0/0
→ 0-bit prefix match

128.143.71.21

Destination address	Next hop
10.0.0.0/8	R1
128.143.0.0/16	R2
128.143.64.0/20	R3
128.143.192.0/20	R3
128.143.71.55/32	R3
default	R5

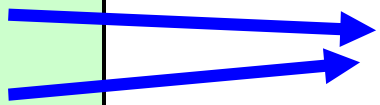
The longest prefix match for 128.143.71.21 is for 24 bits with entry 128.143.71.0/24

Datagram will be sent to R4

Route Aggregation

- Longest prefix match algorithm permits to aggregate prefixes with identical next hop address to a single entry
- This contributes significantly to reducing the size of routing tables of Internet routers

Destination	Next Hop
10.1.0.0/24	R3
10.1.2.0/24	direct
10.2.1.0/24	direct
10.3.1.0/24	R3
20.2.0.0/16	R2
30.1.1.0/28	R2



Destination	Next Hop
10.1.0.0/24	R3
10.1.2.0/24	direct
10.2.1.0/24	direct
10.3.1.0/24	R3
20.0.0.0/8	R2

How do routing tables get updated?

- Adding an interface:
  - Configuring an interface eth2 with 10.0.2.3/24 adds a routing table entry:

Destination	Next Hop/ interface
10.0.2.0/24	eth2

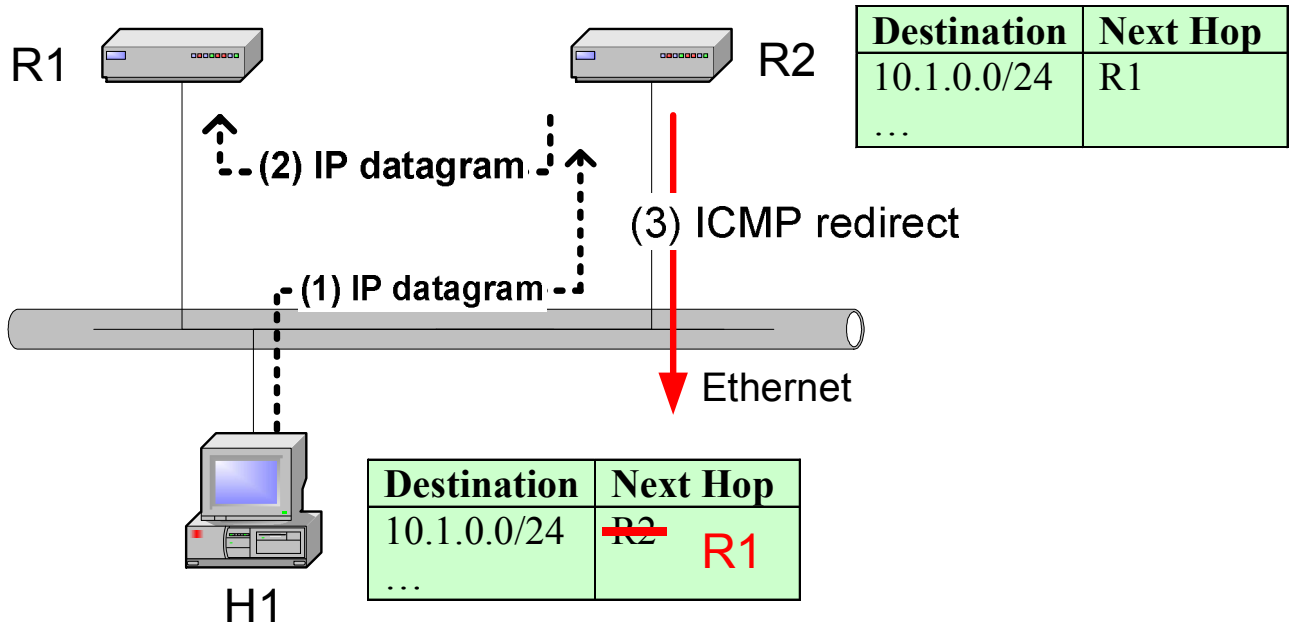
- Adding a default gateway:
  - Configuring 10.0.2.1 as the default gateway adds the entry:

Destination	Next Hop/ interface
0.0.0.0/0	10.0.2.1

- Static configuration of network routes or host routes
- Update of routing tables through routing protocols
- ICMP messages

Routing table manipulations with ICMP

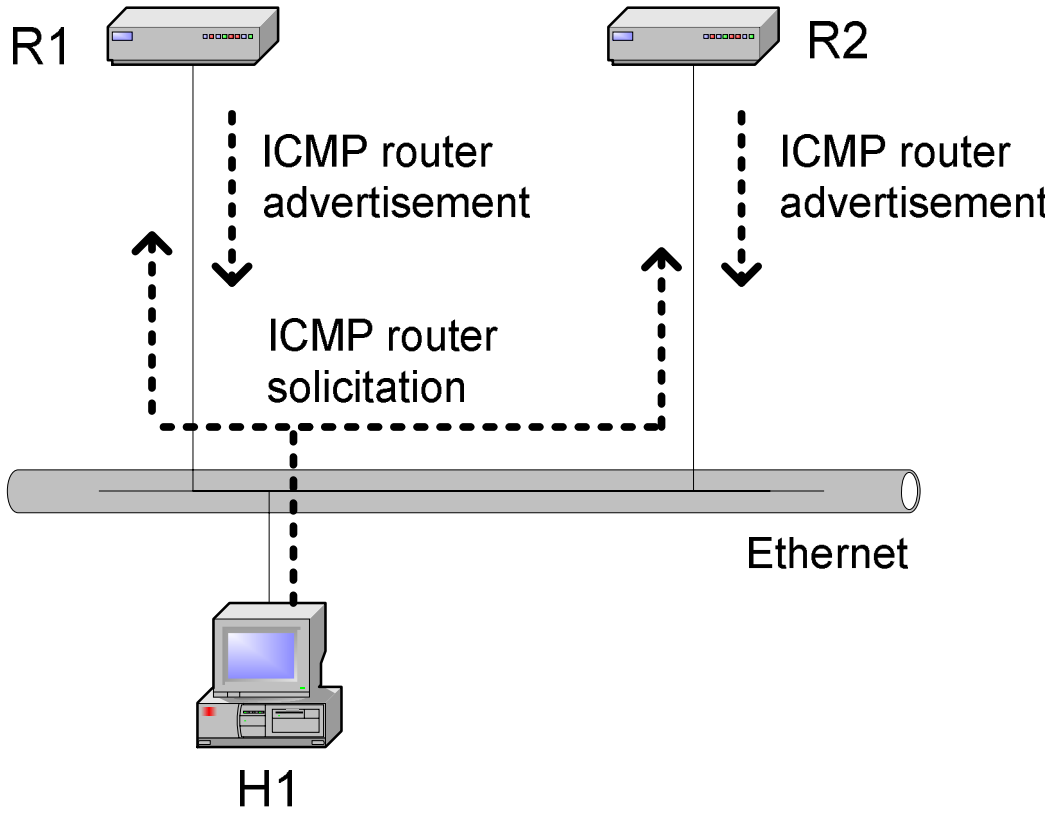
- When a router detects that an IP datagram should have gone to a different router, the router (here R2)
  - forwards the IP datagram to the correct router
  - sends an ICMP redirect message to the host
- Host uses ICMP message to update its routing table



ICMP Router Solicitation
ICMP Router Advertisement

- After bootstrapping a host broadcasts an **ICMP router solicitation**.
- In response, routers send an **ICMP router advertisement** message
- Also, routers periodically broadcast **ICMP router advertisement**

This is sometimes called the Router Discovery Protocol



portscan

- myportscan 192.80.2.10
  - koodi sisse mingi hulk porte skannimiseks
    - pohipordid 80, shh, ftp, telnet,
    - alla saja pordi
- myportscan 192.80.2 8080
  - skannib koik 255 masinat seal all
  - ja kasutab antud pordinumbrit ainult
- argc, argv[0], argv[1], atoi(..), strstr, strchr

- 0..100
- 100.200
- aadressid:
  - mail.startmaster.ru (ip-d jne?) (21,80,...)
  - hamburg.germany.nsu.ru
  - mailserv.monamour.ru
  - mail1.nwpi.ru